Numéro d’inscription: 27958

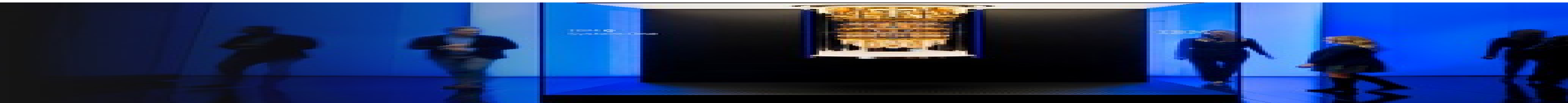
Cryptage et décryptage du chiffrement RSA : Ordinateur classique vs ordinateur quantique

Thème: enjeux sociétaux: environnement, sécurité, énergie

INTRODUCTION

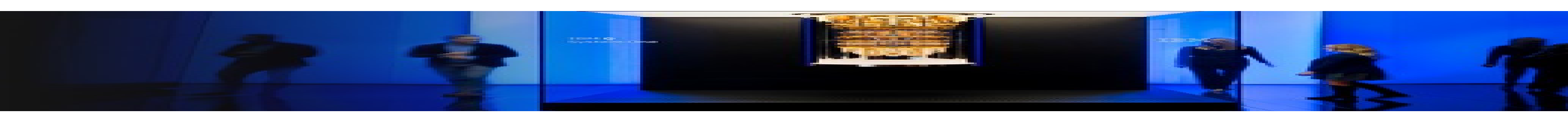
La cybersécurité est devenue l’une des préoccupations majeures des dirigeants d’entreprises puisque les données clients et les informations financières sont toujours menacées par les cyberattaques.
Le constat étant fait, on s’intéressera à étudier le chiffrement RSA, étant le plus utilisé, et envisager quelques méthodes de décryptage classiques et quantiques.

https://www.tice-education.fr/tous-les-articles-er-ressources/articles-informatiques/1392-comprendre-les-principes-de-base-de-la-cryptographie



plan

- Chiffrement RSA
  - Principe
  - Etude comparative de la complexité
- Algorithme de Shor
  - Les bases mathématique et physique de la cryptographie quantique
  - Principe de l'algorithme de Shor
  - Simulation par un vrai calculateur quantique



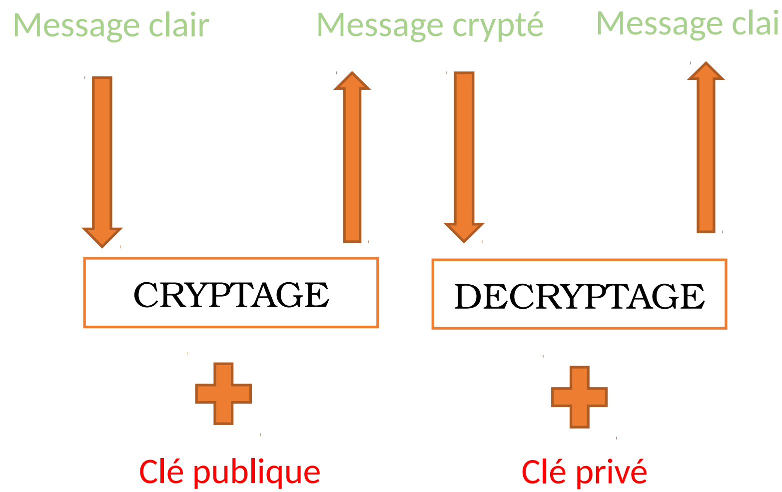
Chiffrement RSA: principe

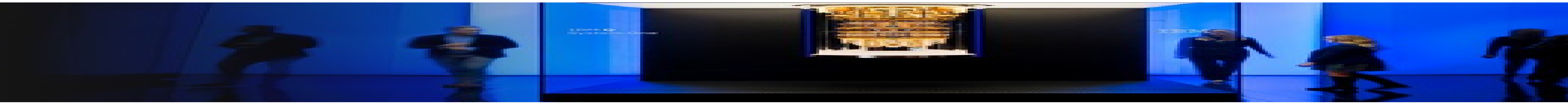
La méthode de cryptage RSA est basé sur le théorème d'Euler :

$$a \wedge n = 1 \Rightarrow a^{\varphi(n)} \equiv 1 \bmod(n)$$

Et sur le résultat suivant :
Soit n=p*q ou p et q premiers. Soit $r \wedge \varphi(n) = 1$ et s tel que
$rs = 1 \bmod(\varphi(n))$ alors $a^{rs} = a \bmod(n)\, pour\ tout\ a \in \mathbb{N}$

En effet: on a $rs = 1 + k\varphi(n)\, et\ \varphi(n) = (p-1)(q-1)$
Pour montrer que $n / a^{rs} - a$ il suffit de démontrer que
$p / a^{rs} - a\, et\ q / a^{rs} - a$
Puisque $p \wedge q = 1$
$si\ a \wedge p > 1\ alors\ p / a\ et\ a^{rs} = a \bmod(p) = 0 \bmod(p)$
Si $a \wedge p = 1$ alors $a^{p-1} = 1 \bmod(p)\, et\ donc\ a^{\varphi(n)} = 1 \bmod(p)$
Donc $a^{rs} = a^{1+k\varphi(n)} = a \bmod(p)$





Implémentation RSA et étude de complexité

- Choisir deux nombres p et q assez grands.
- Calculer n=p*q et m=(p-1)*(q-1)
- Choisir r / 1<r<m et pgcd(r,m)=1
- M le message qu'on veut envoyer sous forme d'un entier .

L'expéditeur

- Envoie le message M^r mod(n)
- n, r et même le message crypté peuvent être publics.

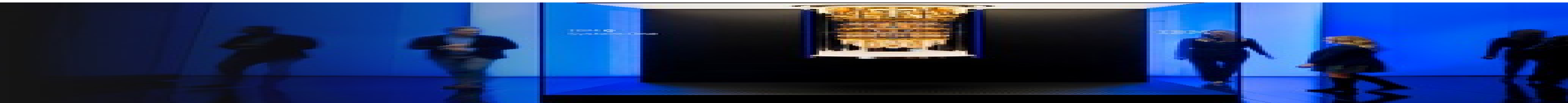
  p, q doivent être privés.

Le receveur

- connait p et q

  Calcule s /

  rs=1 mod(m)
- Calcule M^rs mod(n) et il va retrouver M

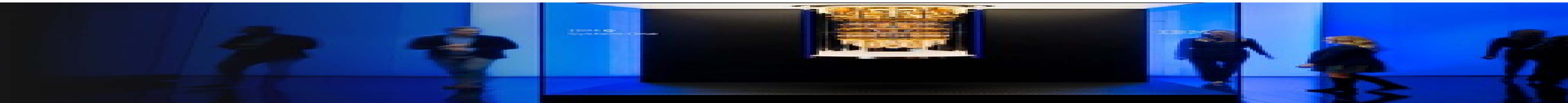


Exemple à la main

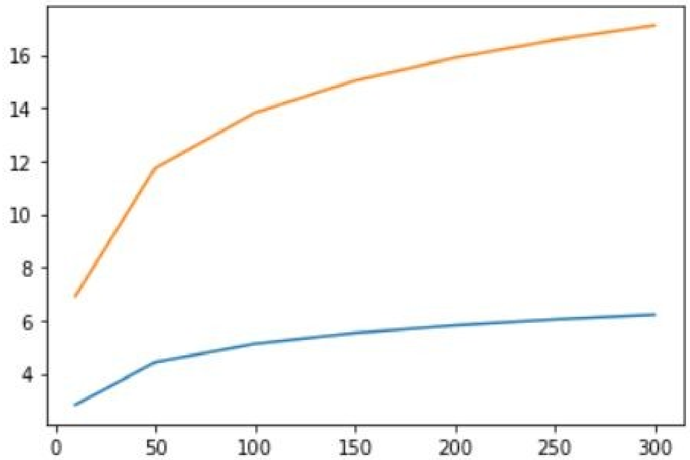
p = 7
q = 19
→ n = 7 * 19 =133
→ m = (p-1) * (q-1) = 6 * 18 = 108
→ Choisir un entier 1<r<m et pgcd(r , m)=1→ r = 5
→ Déterminer un entier s tel que rs=1 mod(m)
→ s=65
→ Clé publique : (n = 133 ; r = 5)
→ Clé privée : (n = 133 ; s = 65)

Supposons qu'on cherche à transmettre x = 6
Cryptage : y = x^r mod(n) = 6^5 mod(133) = 7776 mod 133 = 62
Décryptage : x = y^s mod(n)
= 62^65 mod(133)
= 62 * 120^32 mod(133)
= 62 * 36^16 mod(133) = 62 * 99^8 mod(133)
= 62 * 92^4 mod(133) = 62 * 85^2 mod(133)
= 62 * 43 mod(133) = 2666 mod(133) =6

http://cyberjustice.blog/index.php/2020/02/17/la-cryptographie-quantique-pour-un-monde-non-piratable-de-la-donnee/



Représentations logarithmiques du nombre d'itérations



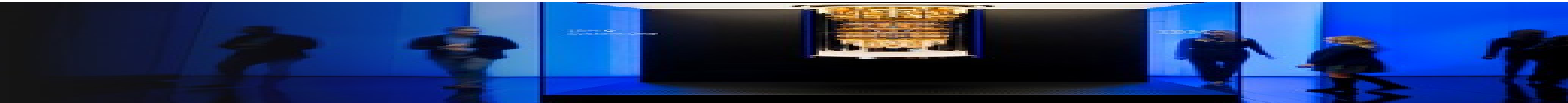
Log(Nombre d'opérations)

Nombre de chiffres

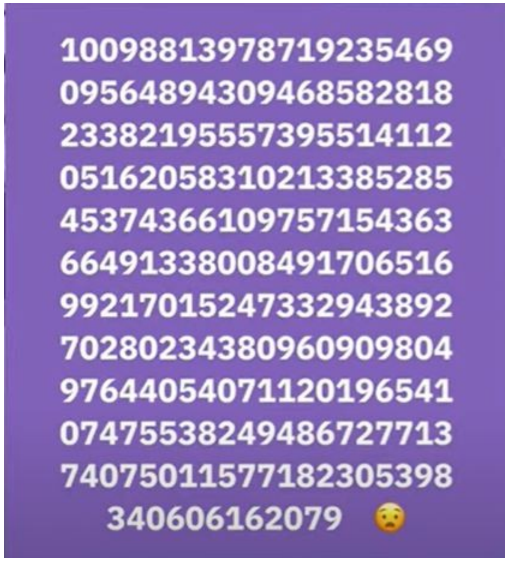
Exp(const*d^1/3)
Meilleur algorithme classique

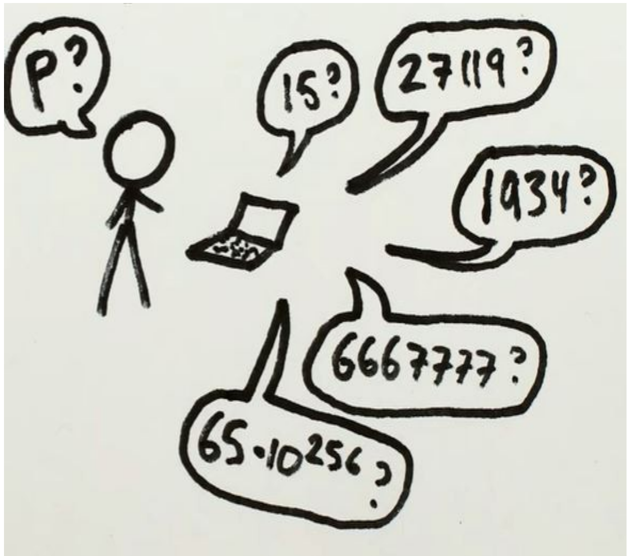
const*d^3
algorithme de Shor

Voir annexe(RSA/Shor/Codes des graphes)

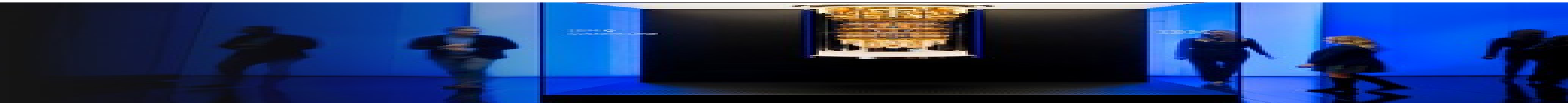


Estimation du temps de calcul





Pour factoriser cet entier, un ordinateur classique a besoin de 1000 années.



L'informatique quantique: les bases mathématiques et physiques

Les outils mathématiques 1/2

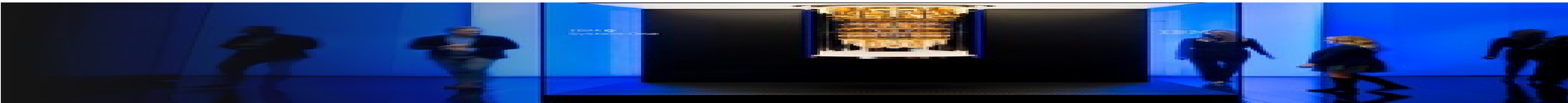
En physique quantique, un système est conçu comme un espace Hilbert. C'est un espace vectoriel muni d'une base dénombrable et un produit scalaire hermitien.

Operateur hermitien

- Un opérateur u d'un espace hermitien E est dit hermitien si: $\forall (x, y) \in E^2,\ (u(x)|y) = (x|u(y))$

$(\overline{A})^T = A$ → Matrice hermitienne

- Diagonalisable(matrice de passage unitaire)
- Valeurs propres réelles
- Les sous espaces propres orthogonaux deux à deux



Les outils mathématiques 2/2

En mécanique quantique, les opérateurs hermitiens représentent les grandeurs physiques.
Les valeurs propres $\Leftrightarrow$ les valeurs possibles de la grandeur
Les vecteurs propres $\Leftrightarrow$ les états associés

Transformation unitaire

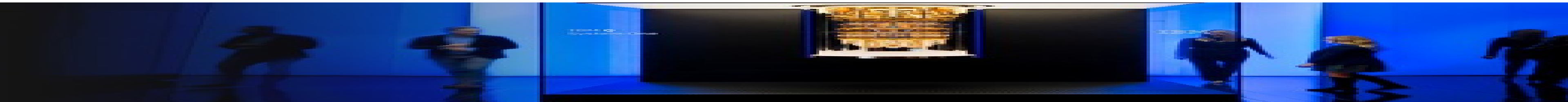
- Une transformation est dite unitaire si: $\forall x, y \in H_1, \quad \langle U(x), U(y)\rangle = \langle x, y\rangle.$

Matrice unitaire $\mathbf{U}^* \times \mathbf{U} = \mathbf{U} \times \mathbf{U}^* = \mathbf{I}$

Matrice de Hadamard

Matrice de Hadamard est une matrice carrée dont les coefficients sont tous 1 ou −1 et dont les lignes sont toutes orthogonales entre elles.

$\Rightarrow$ En mécanique quantique, l'état d'un système est représenté par un vecteur dans un espace de Hilbert.



L'informatique quantique: les bases mathématiques et physiques

La mécanique quantique

1/Les concepts: 1/2

Quantification:

Etat quantique=spectre discontinu

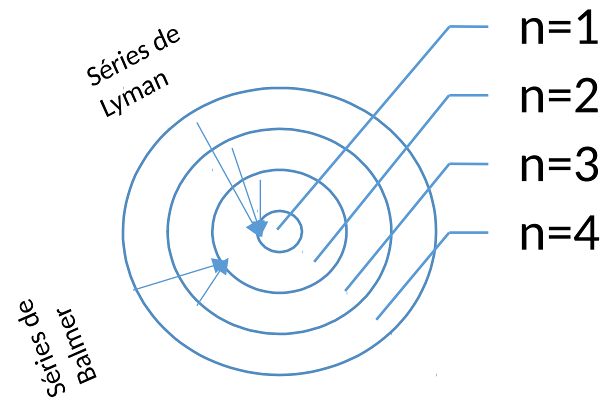
Superposition:

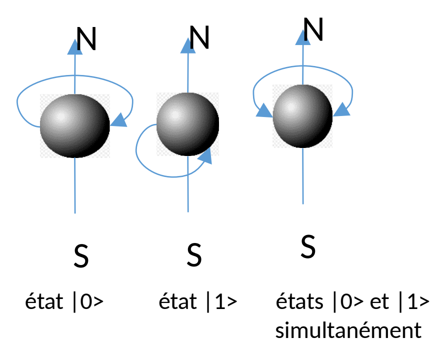
Plusieurs états quantiques existent simultanément.

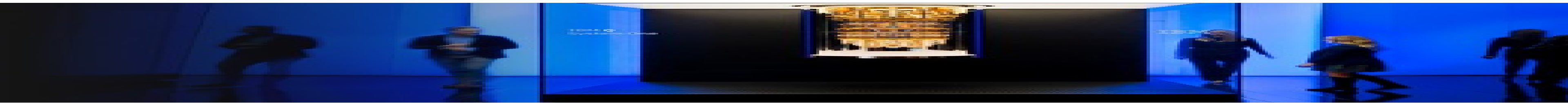
Principe d'incertitude:

On ne peut pas connaitre la position et la quantité de mouvement exacte d'une particule en même temps.

$$\Delta x \Delta p \geq \hbar$$







Dualité onde-particule:

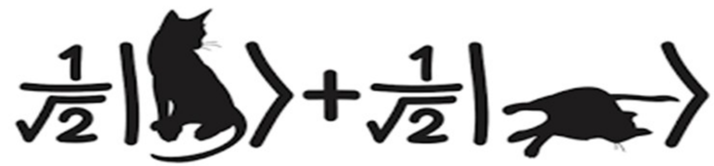
Un quantum est interprété comme onde et particule à la fois.

intrication:

Lorsque deux particules forment un système lié.

Effet tunnel:

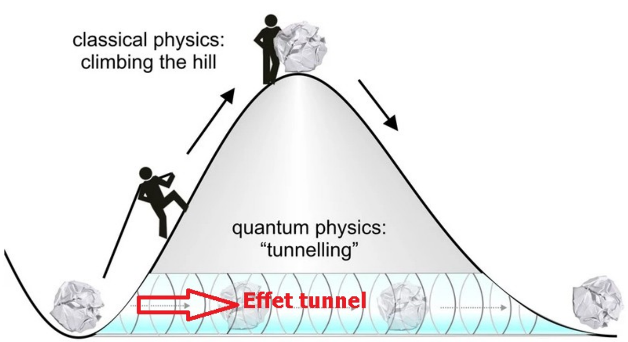
On ne peut pas connaitre la position et la quantité de mouvement exacte d'une particule en même temps.



https://steemit.com/science/@tschelpps/schrodinger-s-cat-vs-the-copenhagen-school-of-thought-quantum-weirdness

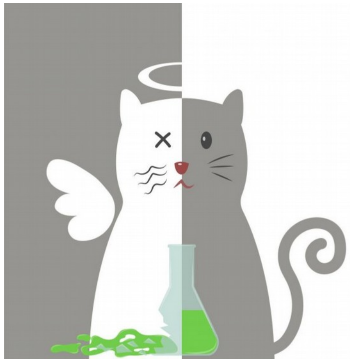
Non-clonage:

Impossibilité de faire des copies identiques d'états quantiques.

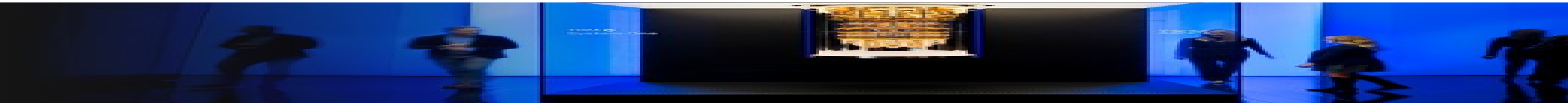


https://www.oezratty.net/wordpress/2018/comprendre-informatique-quantique-basiques/

https://www.shutterstock.com/fr/image-illustration/particle-quantum-entanglement-correlation-3d-illustration-1490864351



https://www.dreamstime.com/illustration/schrodinger-cat.html



2/équation de Schrödinger:

$$-\frac{\hbar}{2m}\nabla^2\psi + V\psi = i\hbar\frac{\partial\psi}{\partial t}$$

La résolution de l'équation de Schrödinger permet de décrire l'évolution dans le temps d'une particule.

La probabilité de présence des particule par un état quantique est $|\psi|^2$

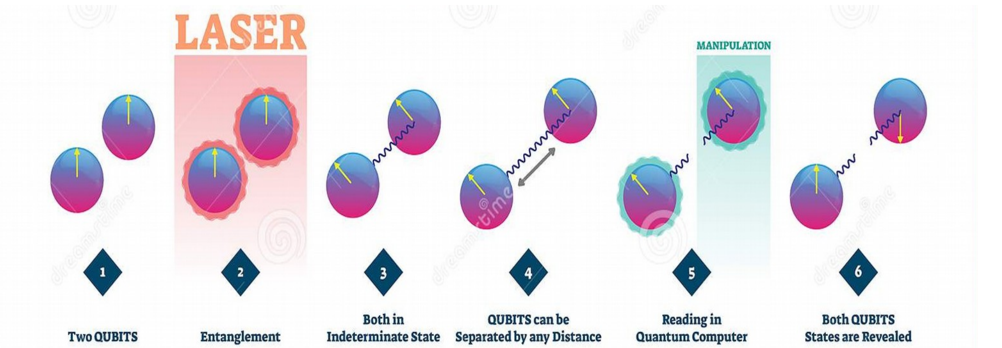
Description d'un état superposé:

$$|\,système\,quantique> = \alpha\,|\,état\,1> + \beta\,|\,état\,2>$$

$|\alpha|^2$ La probabilité de présence dans l'état 1

$|\beta|^2$ La probabilité de présence dans l'état 2

3/intrication des qubits:

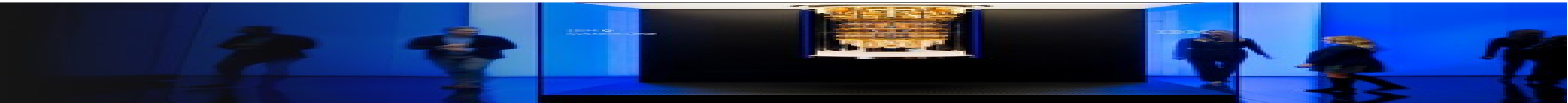


https://www.gograph.com/clipart/qubits-vector-illustration-infographic-with-gg107026014.html

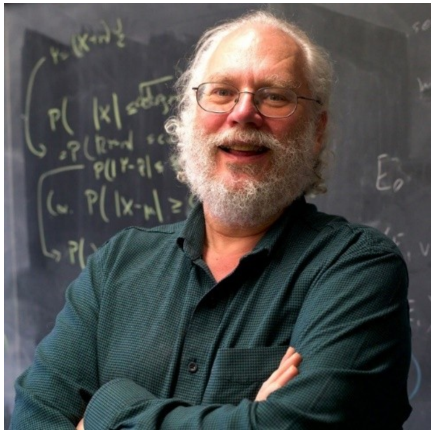
$$|qbit\,1\rangle = \alpha_1|0\rangle + \beta_1|1\rangle$$
$$|qbit\,2\rangle = \alpha_2|0\rangle + \beta_2|1\rangle$$
$$|qbit\,3\rangle = \alpha_3|0\rangle + \beta_3|1\rangle$$

intrication

$$|mémoire\rangle = a|000\rangle + b|001\rangle + c|010\rangle + d|011\rangle + e|100\rangle + f|101\rangle + g|110\rangle + h|111\rangle$$



L'algorithme de Shor: principe



<u>*1/partie classique*</u>*: trouver les facteurs à partir de la*

$(\mathbb{Z}/n\mathbb{Z})^*$ Contient les entiers inferieur à N et premier avec N en muni de modulo N.

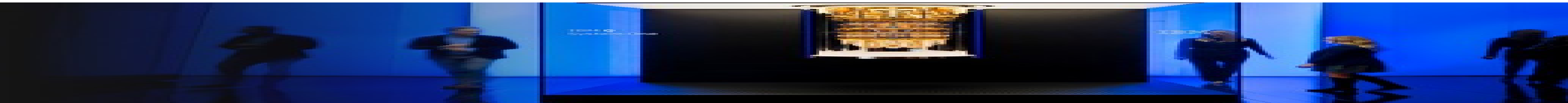
$a \in (\mathbb{Z}/n\mathbb{Z})^*$ $\Rightarrow$ a possède un ordre fini r.

$a^r \equiv 1 \bmod N$ $\Rightarrow$ N| $a^r - 1$

Supposons qu'on peut trouver un entier r pair
Ainsi $N \mid (a^{r/2} - 1)(a^{r/2} + 1)$

Transformée de Fourier:

	Classique	quantique
Formule	$y_k = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x_n \omega_N^{-kn}$	$\mathrm{QFT}: \lvert x\rangle \mapsto \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \omega_N^{xk} \lvert k\rangle.$
Complexité	$O(n2^n)$	$O(n \log n)$

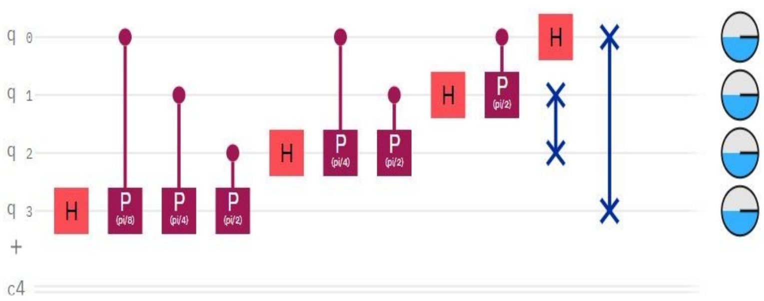


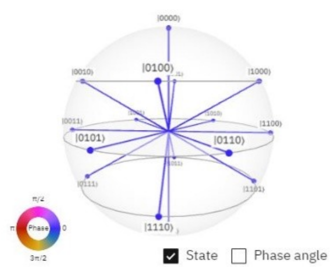
Composition de la transformée de Fourier(QFT) via IBM

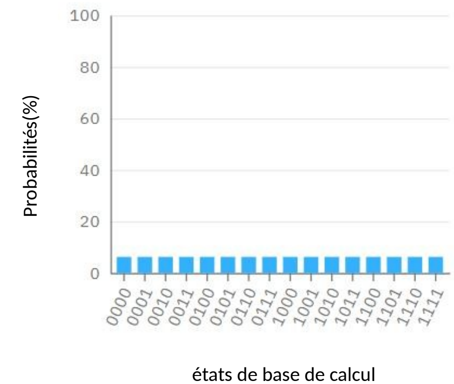
$$|1\rangle + |2\rangle + |4\rangle + |8\rangle + |16\rangle + \ldots$$

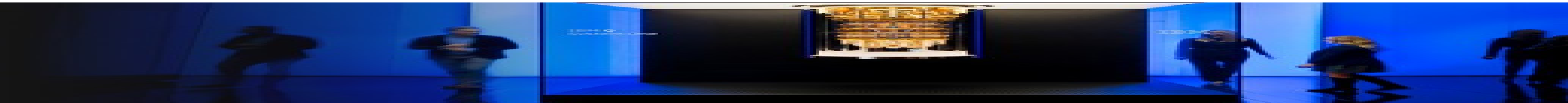
Période

Circuit de 4 bits(IBM)



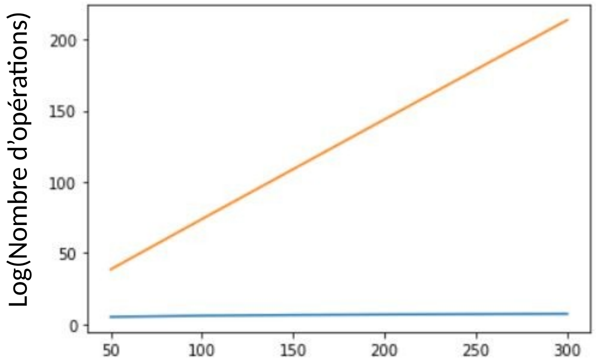






Représentations logarithmiques du nombre d’itérations

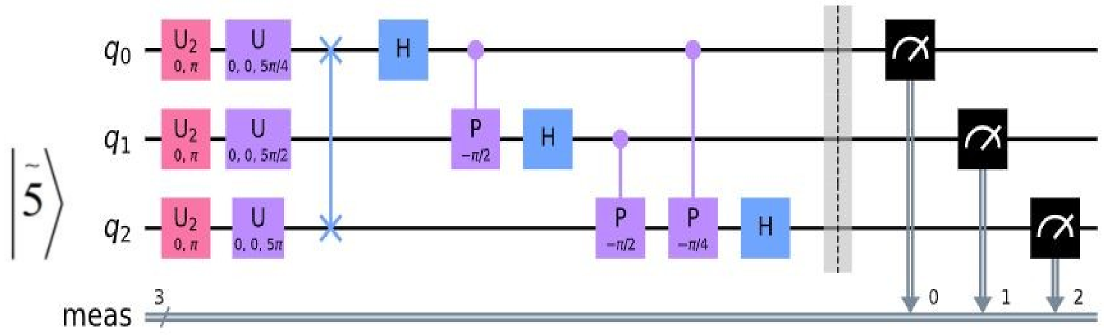
classique



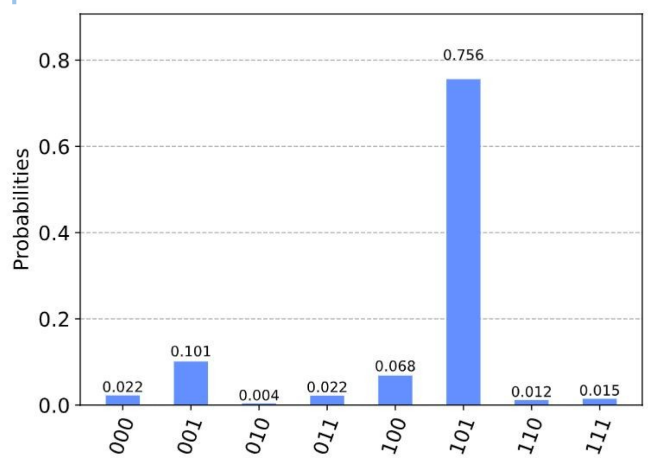
Nombre de chiffres

Voir annexe (codes des graphes)

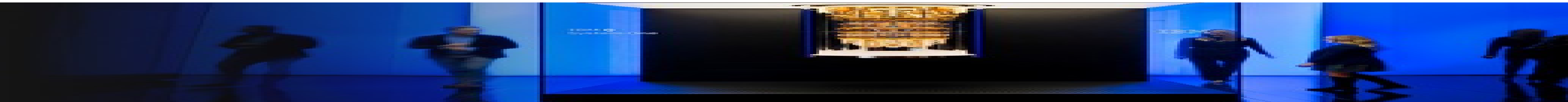
Circuit de la transformée de Fourier quantique inversée (état |5>)



quantique



$$QFT^{\dagger}\left|\tilde{5}\right\rangle = |101\rangle$$



L’algorithme de Shor:

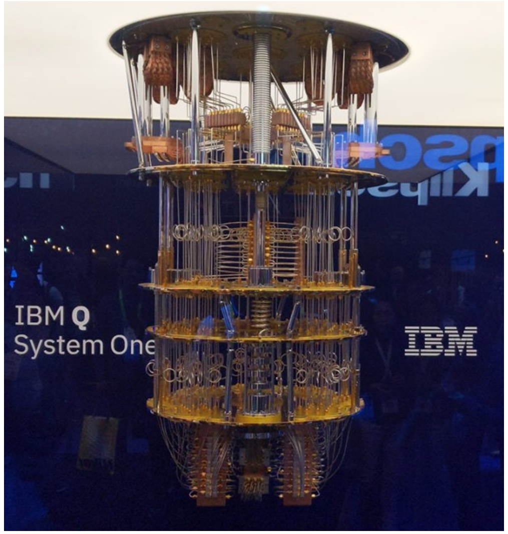
factorisation de 15

```
from qiskit.aqua.algorithms import Shor
from qiskit.aqua import QuantumInstance
import numpy as np
from qiskit import QuantumCircuit, Aer, execute
from qiskit.tools.visualization import plot_histogram
```

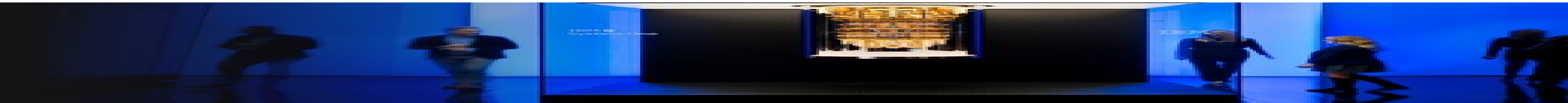
```
backend= Aer.get_backend('qasm_simulator')
quantum_instance=QuantumInstance(backend, shots=1000)
my_shor=Shor(N=15,a=2,quantum_instance=quantum_instance)
```

```
Shor.run(my_shor)
```

```
{'factors': [[3, 5]], 'total_counts': 66, 'successful_counts': 16}
```

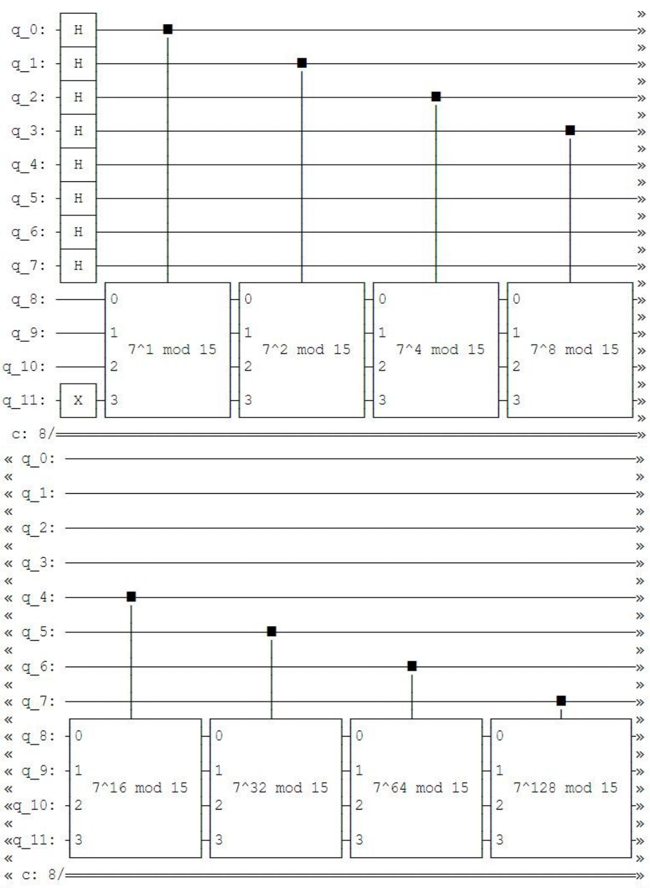


https://www.lebigdata.fr/ibm-q-system-one-ces-2019

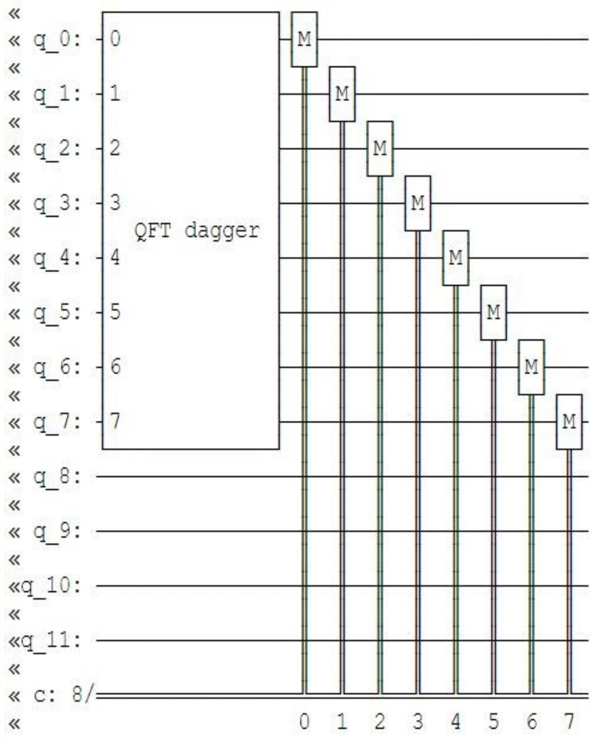


Simulation(n=8,a=7)

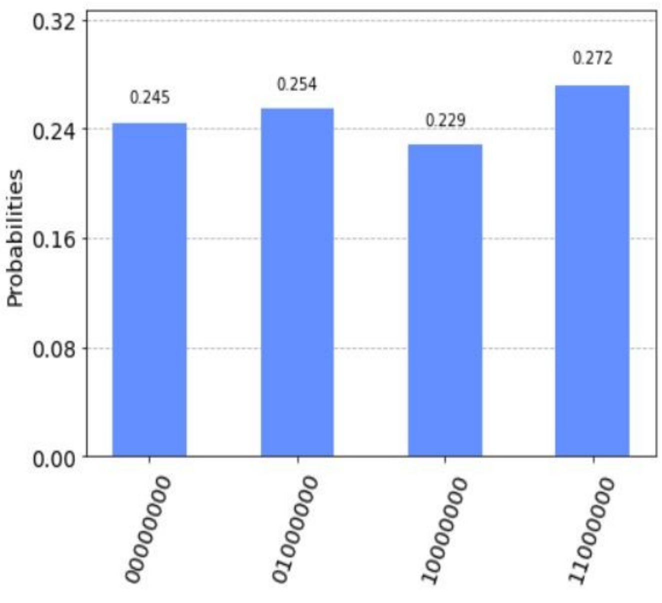
Exponentielle modulaire:



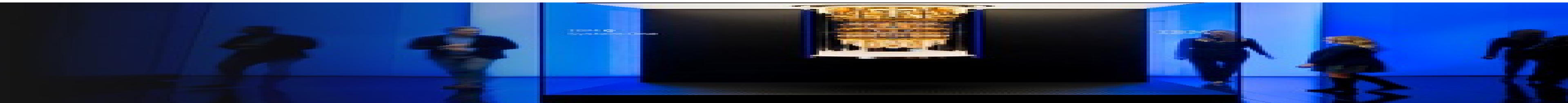
Transformée de Fourier :



Résultats:



Voir annexe page2



Analyse des résultats:

Détermination de la période r & et détermination des facteur de 15

00000000(bin) = 0(dec) 0/256 = 0.00
0/1

01000000(bin) = 64(dec) 64/256 = 0.25
1/4

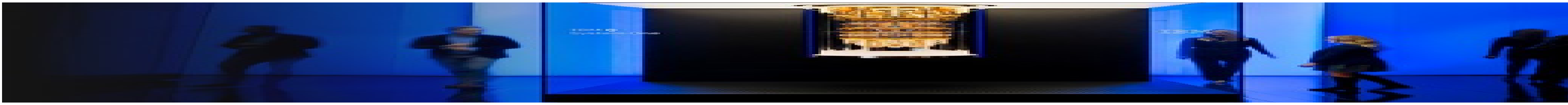
10000000(bin) = 128(dec) 128/256 = 0.50
1/2

11000000(bin) = 192(dec) 192/256 = 0.75
3/4

La valeur la plus probable r=4

$$q = a^{r/2} + 1$$ → q=50 → 5

$$p = a^{r/2} - 1$$ → p=48 → 3



Conclusion

1/

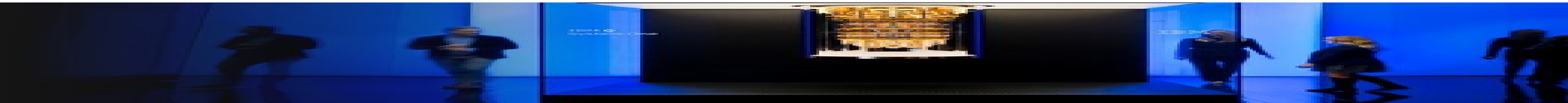
La sécurité de l’algorithme RSA est garantie par sa complexité énorme.

2/

L’algorithme quantique de Shor capable de casser la RSA en un temps polynomial ⇨ la sécurité des données est menacée.

3/

Il est temps d’entamer la cryptographie post-quantique pour faire face au danger de l’ordinateur quantique.



Annexe

RSA

```python
import random
def pgcd(a, b):
    while b != 0:
        a, b = b, a % b
    return a

def multiplicative_inverse(e, phi):
    d = 0
    x1 = 0
    x2 = 1
    y1 = 1
    temp_phi = phi
    while e > 0:
        temp1 = temp_phi/e
        temp2 = temp_phi - temp1 * e
        temp_phi = e
        e = temp2
        x = x2- temp1* x1
        y = d - temp1 * y1
        x2 = x1
        x1 = x
        d = y1
        y1 = y
    if temp_phi == 1:
        return d + phi
```

```python
def est_premier(num):
    if num == 2:
        return True
    if num < 2 or num % 2 == 0:
        return False
    for n in range(3, int(num**0.5)+2, 2):
        if num % n == 0:
            return False
    return True

def generer_cle(p, q):
    if not (est_premier(p) and est_premier(q)):
        raise ValueError('Both numbers must be prime.')
    elif p == q:
        raise ValueError('p and q cannot be equal')
    n = p * q
    phi = (p-1) * (q-1)
    e = random.randrange(1, phi)
    g = pgcd(e, phi)
    while g != 1:
        e = random.randrange(1, phi)
        g = pgcd(e, phi)
    d = multiplicative_inverse(e, phi)
    return ((e, n), (d, n))
```

```python
def encrypt(pk, plaintext):
     key, n = pk
     cipher = [(ord(char) ** key) % n for char in plaintext]
     return cipher

def decrypt(pk, ciphertext):
    key, n = pk
    plain = [chr((char ** key) % n) for char in ciphertext]
    return ''.join(plain)
```

```python
print ("RSA Encrypter/ Decrypter")
p = int(input("saisir un nombre premier (17, 19, 23, etc): "))
q = int(input("saisir un nombre premier différent de celui choisi au dessus: "))
print ("génération de la clé public / privée  . . .")
public, private = generer_cle(p, q)
print ("la clé public est "), public ,(" et la clé privée est "), private
message = input("entrer le message à encrypter: ")
encrypted_msg = encrypt(private, message)
print ("votre message encrypté est : ")
print (''.join(map(lambda x: str(x), encrypted_msg)))
print ("décrypter le message "), public ,(" . . .")
print ("votre message est :")
print (decrypt(public, encrypted_msg))
```

Algorithme de Shor

```python
def c_amod15(a, power):
    if a not in [2,7,8,11,13]:
        raise ValueError("'a' must be 2,7,8,11 or 13")
    U = QuantumCircuit(4)
    for iteration in range(power):
        if a in [2,13]:
            U.swap(0,1)
            U.swap(1,2)
            U.swap(2,3)
        if a in [7,8]:
            U.swap(2,3)
            U.swap(1,2)
            U.swap(0,1)
        if a == 11:
            U.swap(1,3)
            U.swap(0,2)
        if a in [7,11,13]:
            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
```

```python
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('text')
```

```python
backend = Aer.get_backend('qasm_simulator')
results = execute(qc, backend, shots =2048).result()
counts = results.get_counts()
plot_histogram(counts)
```

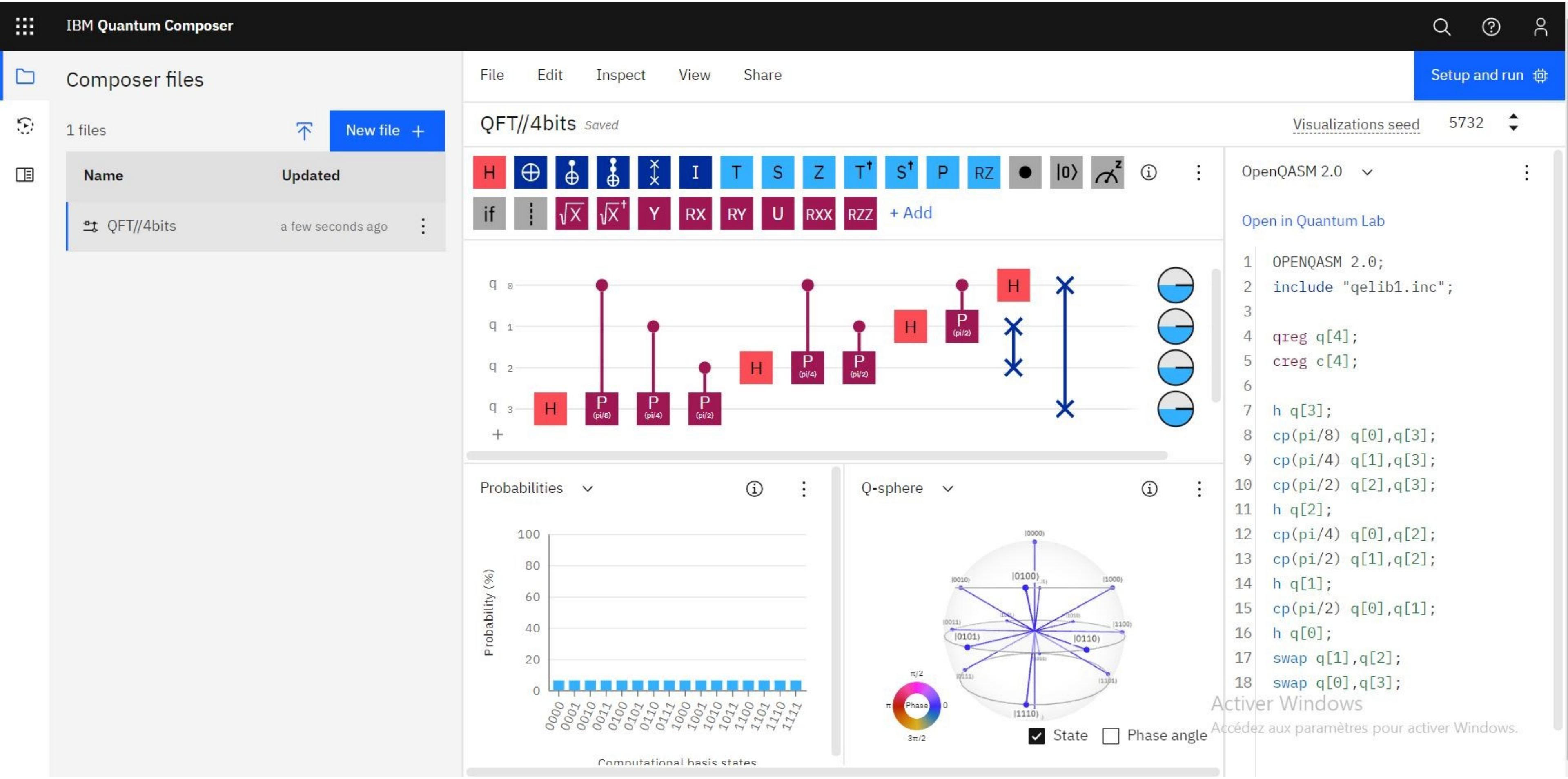
```python
a = 7
factor_found = False
attempt = 0
while not factor_found:
    attempt += 1
    print("\nAttempt %i:" % attempt)
    phase = qpe_amod15(a) # Phase = s/r
    frac = Fraction(phase).limit_denominator(15)
    r = frac.denominator
    print("Result: r = %i" % r)
    if phase != 0:
        guesses = [gcd(a**(r//2)-1, 15), gcd(a**(r//2)+1, 15)]
        print("Guessed Factors: %i and %i" % (guesses[0], guesses[1]))
        for guess in guesses:
            if guess != 1 and (15 % guess) == 0: # Check to see if guess is a factor
                print("*** Non-trivial factor found: %i ***" % guess)
                factor_found = True
```

```python
def qft_dagger(n):
    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 dagger"
    return qc
```

```python
def a2jmodN(a, j, N):
    for i in range(j):
        a = np.mod(a**2, N)
    return a

def qpe_amod15(a):
    n_count = 3
    qc = QuantumCircuit(4+n_count, 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))
    backend = Aer.get_backend('qasm_simulator')
    result = execute(qc, backend, shots=1, memory=True).result()
    readings = result.get_memory()
    print("Register Reading: " + readings[0])
    phase = int(readings[0],2)/(2**n_count)
    print("Corresponding Phase: %f" % phase)
    return phase
```

Création du circuit de la QFT à 4 bits via l'expérience quantique IBM



Codes python des graphes de complexité

```
import matplotlib.pyplot as plt
import numpy as np
```

```
import numpy as np
X=[10]
for i in range(1,7):
    X.append(i*50)
f1=lambda d: np.log((d**1/3))
f2=lambda d: np.log(d**3)
Y1=[f1(K) for K in X]
Y2=[f2(K) for K in X]
plt.plot(X,Y1)
plt.plot(X,Y2)
plt.show()
```

```
import matplotlib.pyplot as plt
import numpy as np
X=[]
for i in range(1,7):
    X.append(i*50)
f1=lambda d: np.log((d*np.log(d)))
f2=lambda d: np.log(d)+d*np.log(2)
Y1=[f1(K) for K in X]
Y2=[f2(K) for K in X]
plt.plot(X,Y1)
plt.plot(X,Y2)
plt.show()
```