Introduction to Quantum Computing – Quantum Applications

Marcelo Ponce

June 8, 2023

CMS-UTSC/SciNet

Today's lecture

The goal for this session is to discuss how some applications/algorithms for quantum computers work and can be implemented.

We will discuss the following topics:

- Shor's Algorithm

  Modular arithmetics
  Classical Factorization
  Quantum Phase Estimation
  Period Estimation (QFT)
  RSA Basics

- Post Quantum Cryptography
- Concluding remarks

Today's lecture

The goal for this session is to discuss how some applications/algorithms for quantum computers work and can be implemented.

We will discuss the following topics:

- Shor's Algorithm

  Modular arithmetics Quantum Phase Estimation RSA Basics

  Classical Factorization Period Estimation (QFT)

- Post Quantum Cryptography
- Concluding remarks

Material based on Xanadu's codebook and PennyLane documentation.

Please stop me if you have any questions or doubts.

Shor’s Algorihtm

Shor's Algorithm

- One of the most famous algorithms in quantum computing.
- Originally developed in 1994 by Peter Shor, way before a QC was even on a blue print.
- It is a QC algorithm for finding the prime factors of an integer.
- It mixes classical and quantum procedures to solve the problem of factoring numbers.
- This posses serious concerns, basically breaks our typical cryptographic techniques.
- In particular, the RSA system is based on the fact that factorizing large numbers cannot be achieved efficiently.

Modular Arithmetics

Modular arithmetic works with integers $\mathbb{Z}$

Just like traditional arithmetic, we can perform addition, subtraction, and multiplication using modular arithmetic.

Modular Arithmetics

Modular arithmetic works with integers $\mathbb{Z}$

Just like traditional arithmetic, we can perform addition, subtraction, and multiplication using modular arithmetic.

The fundamental difference is that we group the numbers according to their remainder when dividing by an integer m.

I.e. given a number m, we will say that $a \equiv b \pmod{m}$ if $a - b$ is an integer multiple of m

Modular Arithmetics

Modular arithmetic works with integers $\mathbb{Z}$

Just like traditional arithmetic, we can perform addition, subtraction, and multiplication using modular arithmetic.

The fundamental difference is that we group the numbers according to their remainder when dividing by an integer m.

I.e. given a number m, we will say that $a \equiv b \pmod{m}$ if $a - b$ is an integer multiple of m

Read as "a is equivalent to b modulo m" (or, "mod m").

E.g. $13 \equiv 3 \pmod{5}$, since $13 - 3 = 10$ and and 10 is a multiple of 5.

Modular Arithmetics

Modular arithmetic works with integers $\mathbb{Z}$

Just like traditional arithmetic, we can perform addition, subtraction, and multiplication using modular arithmetic.

The fundamental difference is that we group the numbers according to their remainder when dividing by an integer m.

I.e. given a number m, we will say that $a \equiv b \pmod{m}$ if $a - b$ is an integer multiple of m

Read as "a is equivalent to b modulo m" (or, "mod m").

E.g. $13 \equiv 3 \pmod{5}$, since $13 - 3 = 10$ and and 10 is a multiple of 5.

Basic Arithmetics

Within modular arithmetic, the equivalence is maintained after addition, multiplication, or exponentiation.

Modular Arithmetics

Modular arithmetic works with integers $\mathbb{Z}$

Just like traditional arithmetic, we can perform addition, subtraction, and multiplication using modular arithmetic.

The fundamental difference is that we group the numbers according to their remainder when dividing by an integer m.

I.e. given a number m, we will say that $a \equiv b \pmod{m}$ if $a - b$ is an integer multiple of m

Read as "a is equivalent to b modulo m" (or, "mod m").

E.g. $13 \equiv 3 \pmod{5}$, since $13 - 3 = 10$ and and 10 is a multiple of 5.

Basic Arithmetics

Within modular arithmetic, the equivalence is maintained after addition, multiplication, or exponentiation.

$a \equiv b \pmod{m} \Rightarrow a + n \equiv b + n \pmod{m}, \forall n \in \mathbb{Z}$

$a \equiv b \pmod{m} \Rightarrow a \cdot n \equiv b \cdot n \pmod{m}, \forall n \in \mathbb{Z}$

$a \equiv b \pmod{m} \Rightarrow a^n \equiv b^n \pmod{m}, \forall n \in \mathbb{Z}^+$

Inverse of a number
An integer c is inverse of a modulo m if $ac \equiv 1 \pmod{m}$.

For example, the inverse of 5 modulo 7 is 3 since $5 \cdot 3 \equiv 15 \pmod{7}$.

However, not all numbers have inverses since, for example, if we work with modulo 15, there is no integer c such that $5c \equiv 1 \pmod{15}$.

The numbers that do not have an inverse are those that are not coprime with the modulus m, that is, if a and m have some common factor then a will not have an inverse.

Classical Factorization – Non-trivial square root

Nontrivial square root of m
Defined as that number x which satisfies, $x^2 = 1 \pmod{m}$

where x is an integer such that $x \in \{2, m-2\}$

For example, 5 is a nontrivial square root of 12. That is because $5^2 \equiv 25 \equiv 1 \pmod{12}$.

This nontrivial square root will always exist and does not necessarily have to be unique.

Also, $x = x^{-1} \pmod{m}$, $x \neq \pm 1$

Classical Factorization – Non-trivial square root

Nontrivial square root of m
Defined as that number x which satisfies, $x^2 = 1 \pmod{m}$

where x is an integer such that $x \in \{2, m-2\}$

For example, 5 is a nontrivial square root of 12. That is because $5^2 \equiv 25 \equiv 1 \pmod{12}$.

This nontrivial square root will always exist and does not necessarily have to be unique.

Also, $x = x^{-1} \pmod{m}$, $x \neq \pm 1$

Finding a nontrivial square root is the key to the decomposition of a number

$$x^2 \equiv 1 \pmod{N} \Rightarrow x^2 - 1 \equiv 0 \pmod{N}$$

$$\Rightarrow (x-1)(x+1) \equiv 0 \pmod{N}$$

Classical Factorization – Non-trivial square root

Nontrivial square root of m
Defined as that number x which satisfies, $x^2 = 1 \pmod{m}$

where x is an integer such that $x \in \{2, m-2\}$

For example, 5 is a nontrivial square root of 12. That is because $5^2 \equiv 25 \equiv 1 \pmod{12}$.

This nontrivial square root will always exist and does not necessarily have to be unique.

Also, $x = x^{-1} \pmod{m}$, $x \neq \pm 1$

Finding a nontrivial square root is the key to the decomposition of a number

$x^2 \equiv 1 \pmod{N} \Rightarrow x^2 - 1 \equiv 0 \pmod{N}$
$\Rightarrow (x-1)(x+1) \equiv 0 \pmod{N}$

Supposing that N can be factored as the product of two primes p and q

$\Rightarrow (x-1)(x+1) = kN = kpq$

Classical Factorization – Non-trivial square root

Nontrivial square root of m
Defined as that number x which satisfies, $x^2 = 1 \pmod{m}$

where x is an integer such that $x \in \{2, m-2\}$

For example, 5 is a nontrivial square root of 12. That is because $5^2 \equiv 25 \equiv 1 \pmod{12}$.

This nontrivial square root will always exist and does not necessarily have to be unique.

Also, $x = x^{-1} \pmod{m}$, $x \neq \pm 1$

Finding a nontrivial square root is the key to the decomposition of a number

$x^2 \equiv 1 \pmod{N} \Rightarrow x^2 - 1 \equiv 0 \pmod{N}$
$\Rightarrow (x-1)(x+1) \equiv 0 \pmod{N}$

Supposing that N can be factored as the product of two primes p and q
$\Rightarrow (x-1)(x+1) = kN = kpq$

Let N be an integer that is a product of two primes p and q. If x is a nontrivial square root of N such that $(x-1)(x+1) = kN$ for some integer k, then $(x-1)$ is a multiple of one of the primes and $(x+1)$ is a multiple of the other.

Classical Factorization – GCD

Given two integers, the **Greatest Common Denominator (GCD)** is the largest possible product using the common prime factors.

E.g. $\boldsymbol{GCD(150, 250) = 50}$

Classical Factorization – GCD

Given two integers, the **Greatest Common Denominator (GCD)** is the largest possible product using the common prime factors.

E.g. $GCD(150, 250) = 50$

$150 = 5 \cdot 5 \cdot 3 \cdot 2$

$250 = 5 \cdot 5 \cdot 5 \cdot 5 \cdot 2$

$\Rightarrow 5 \cdot 5 \cdot 2 = 50$

Classical Factorization – GCD

Given two integers, the **Greatest Common Denominator (GCD)** is the largest possible product using the common prime factors.

E.g. $\boldsymbol{GCD(150, 250) = 50}$

$\mathbf{150 = 5 \cdot 5 \cdot 3 \cdot 2}$

$\mathbf{250 = 5 \cdot 5 \cdot 5 \cdot 5 \cdot 2}$

$\Rightarrow \mathbf{5 \cdot 5 \cdot 2 = 50}$

In general, this can be efficiently implemented in a classical way through what is known as *Euclid's algorithm*.

Classical Factorization – GCD

Given two integers, the **Greatest Common Denominator (GCD)** is the largest possible product using the common prime factors.

E.g. $GCD(150, 250) = 50$

$150 = 5 \cdot 5 \cdot 3 \cdot 2$

$250 = 5 \cdot 5 \cdot 5 \cdot 5 \cdot 2$

$\Rightarrow 5 \cdot 5 \cdot 2 = 50$

In general, this can be efficiently implemented in a classical way through what is known as *Euclid's algorithm*.

So, we know that $N = pq$ and $(x+1) = sp$ with s being an integer whose decomposition will not depend on q.

Classical Factorization – GCD

Given two integers, the **Greatest Common Denominator (GCD)** is the largest possible product using the common prime factors.

E.g. $GCD(150, 250) = 50$

$150 = 5 \cdot 5 \cdot 3 \cdot 2$

$250 = 5 \cdot 5 \cdot 5 \cdot 5 \cdot 2$

$\Rightarrow 5 \cdot 5 \cdot 2 = 50$

In general, this can be efficiently implemented in a classical way through what is known as *Euclid's algorithm*.

So, we know that $N = pq$ and $(x + 1) = sp$ with s being an integer whose decomposition will not depend on q.

$$\boxed{\begin{aligned} p &= GCD((x-1), N) \\ q &= GCD((x+1), N) \end{aligned}}$$

Classical Factorization – GCD

Given two integers, the **Greatest Common Denominator (GCD)** is the largest possible product using the common prime factors.

E.g. $GCD(150, 250) = 50$

$150 = 5 \cdot 5 \cdot 3 \cdot 2$

$250 = 5 \cdot 5 \cdot 5 \cdot 5 \cdot 2$

$\Rightarrow 5 \cdot 5 \cdot 2 = 50$

In general, this can be efficiently implemented in a classical way through what is known as *Euclid's algorithm*.

So, we know that $N = pq$ and $(x+1) = sp$ with s being an integer whose decomposition will not depend on q.

$$\boxed{\begin{aligned} p &= GCD((x-1), N) \\ q &= GCD((x+1), N) \end{aligned}}$$

E.g. $2^4 \equiv 1$ (mod 15)

$2^4 = (2^2)^2 \equiv 1$ (mod 15)

$$\Rightarrow \begin{array}{l} p = GCD(2^2 - 1, 15) = 3 \\ 1 = GCD(2^2 + 1, 15) = 5 \end{array}$$

Hands-On: Modular Arithmetic

Implement a function which, given three arguments a, b and m, tells us whether $a \equiv b \pmod{m}$

Hands-On: Modular Arithmetic

Implement a function which, given three arguments a, b and m, tells us whether $a \equiv b \pmod{m}$

```python
def is_equivalent(a, b, m):
    """Return a boolean indicating whether the equivalence is satisfied.

    Args:
        a (int): First number to check the equivalence.
        b (int): Second number to check the equivalence.
        m (int): Modulus of the equivalence.

    Returns:
        bool: True if a = b (m), False otherwise.
    """
```

Hands-On: Modular Arithmetic

Implement a function which, given three arguments a, b and m, tells us whether $a \equiv b \pmod{m}$

```python
def is_equivalent(a, b, m):
    """Return a boolean indicating whether the equivalence is satisfied.

    Args:
        a (int): First number to check the equivalence.
        b (int): Second number to check the equivalence.
        m (int): Modulus of the equivalence.

    Returns:
        bool: True if a = b (m), False otherwise.
    """
```

```python
    return ((a-b) % m == 0)


print(f"13 = 8 (3) is {is_equivalent(13, 8, 3)}")
print(f"13 = 7 (6) is {is_equivalent(13, 7, 6)}")
```

```
13 = 8 (3) is False
13 = 7 (6) is True
```

Hands-On: Modular Inverse

Implement a function in which given a and m, indicates if there is an inverse for the number a modulo m.

Hands-On: Modular Inverse

Implement a function in which given a and m, indicates if there is an inverse for the number a modulo m.

```python
def has_inverse(a, m):
    """Returns a boolean indicating whether a number has an inverse modulo m.

    Args:
        a (int): Number to find the inverse modulus m.
        m (int): Modulus of the equivalence.

    Returns:
        bool: True if c exists (ac = 1 (m)), False otherwise
    """
```

Hands-On: Modular Inverse

Implement a function in which given a and m, indicates if there is an inverse for the number a modulo m.

```python
def has_inverse(a, m):
    """Returns a boolean indicating whether a number has an inverse modulo m.

    Args:
        a (int): Number to find the inverse modulus m.
        m (int): Modulus of the equivalence.

    Returns:
        bool: True if c exists (ac = 1 (m)), False otherwise
    """
```

```python
    for c in range(a):
        if ( (a*c-1) % m == 0 ):
            return True
    return False


print("(5,15)", has_inverse(5,15))
print("(7,15)", has_inverse(7,15))
```

```
(5,15) False
(7,15) False
```

Period Finding

Factorizing a number $=$ $\underbrace{\textbf{non-trivial square roots}}_{\text{quantum alg.}}$ $+$ determine prime factors

Period Finding
the purpose of this algorithm is, given an operator U and a state $|\psi\rangle$, to find an integer r such that: $U^r |\psi\rangle = |\psi\rangle$

the period depends on both the operator and the chosen state.

$r \rightsquigarrow$ period, $U^{r+m} |\psi\rangle = U^m |\psi\rangle, \forall m \in \mathbb{Z}$

Finding is one of those problems that can efficeintly by tackled with a quantum computer.

Period Finding Alg.

Let's assume that we have an state $|\boldsymbol{\Psi_0}\rangle = \frac{1}{\sqrt{r}}(|\phi\rangle + U\,|\phi\rangle + \cdots + U^{r-1}\,|\phi\rangle)$

Period Finding Alg.

Let's assume that we have an state $|\Psi_0\rangle = \frac{1}{\sqrt{r}}(|\phi\rangle + U|\phi\rangle + \cdots + U^{r-1}|\phi\rangle)$

Applying U to $|\Psi\rangle$,

Period Finding Alg.

Let's assume that we have an state $|\boldsymbol{\Psi_0}\rangle = \frac{\boldsymbol{1}}{\sqrt{\boldsymbol{r}}}(|\boldsymbol{\phi}\rangle + \boldsymbol{U}|\boldsymbol{\phi}\rangle + \cdots + \boldsymbol{U}^{\boldsymbol{r-1}}|\boldsymbol{\phi}\rangle)$

Applying $\boldsymbol{U}$ to $|\boldsymbol{\Psi}\rangle$, $\boldsymbol{U}|\boldsymbol{\Psi_0}\rangle = \frac{\boldsymbol{1}}{\sqrt{\boldsymbol{r}}}(\boldsymbol{U}|\boldsymbol{\phi}\rangle + \boldsymbol{U}^{\boldsymbol{2}}|\boldsymbol{\phi}\rangle + \cdots + \boldsymbol{U}^{\boldsymbol{r}}|\boldsymbol{\phi}\rangle)$

Period Finding Alg.

Let's assume that we have an state $|\Psi_0\rangle = \frac{1}{\sqrt{r}}(|\phi\rangle + U|\phi\rangle + \cdots + U^{r-1}|\phi\rangle)$

Applying U to $|\Psi\rangle$, $U|\Psi_0\rangle = \frac{1}{\sqrt{r}}(U|\phi\rangle + U^2|\phi\rangle + \cdots + U^r|\phi\rangle)$

Considering that $U^r|\phi\rangle = |\phi\rangle$, then we get
$U|\Psi_0\rangle = \frac{1}{\sqrt{r}}(U|\phi\rangle + U^2|\phi\rangle + \cdots + |\phi\rangle) = |\Psi_0\rangle$

Period Finding Alg.

Let's assume that we have an state $|\Psi_0\rangle = \frac{1}{\sqrt{r}}(|\phi\rangle + U|\phi\rangle + \cdots + U^{r-1}|\phi\rangle)$

Applying U to $|\Psi\rangle$, $U|\Psi_0\rangle = \frac{1}{\sqrt{r}}(U|\phi\rangle + U^2|\phi\rangle + \cdots + U^r|\phi\rangle)$

Considering that $U^r|\phi\rangle = |\phi\rangle$, then we get

$U|\Psi_0\rangle = \frac{1}{\sqrt{r}}(U|\phi\rangle + U^2|\phi\rangle + \cdots + |\phi\rangle) = |\Psi_0\rangle$

$\therefore$ $|\Psi_0\rangle$ is an eigenvector of U with assoc. eigenvalue 1.

Period Finding Alg.

Let's assume that we have an state $|\Psi_0\rangle = \frac{1}{\sqrt{r}}(|\phi\rangle + U|\phi\rangle + \cdots + U^{r-1}|\phi\rangle)$

Applying U to $|\Psi\rangle$, $U|\Psi_0\rangle = \frac{1}{\sqrt{r}}(U|\phi\rangle + U^2|\phi\rangle + \cdots + U^r|\phi\rangle)$

Considering that $U^r|\phi\rangle = |\phi\rangle$, then we get
$U|\Psi_0\rangle = \frac{1}{\sqrt{r}}(U|\phi\rangle + U^2|\phi\rangle + \cdots + |\phi\rangle) = |\Psi_0\rangle$

$\therefore$ $|\Psi_0\rangle$ is an eigenvector of U with assoc. eigenvalue 1.

Considering, $|\Psi_1\rangle = \frac{1}{\sqrt{r}}(|\phi\rangle + \exp\left[\frac{-2\pi i}{r}\right]U|\phi\rangle + |\phi\rangle +$
$\exp\left[\frac{-2\pi i}{r} \times 2\right]U^2|\phi\rangle + \cdots + \exp\left[\frac{-2\pi i}{r} \times (r-1)\right]U^{r-1}|\phi\rangle)$

i.e. adding an additional phase in each ot the terms.

Period Finding Alg.

Let's assume that we have an state $|\Psi_0\rangle = \frac{1}{\sqrt{r}}(|\phi\rangle + U|\phi\rangle + \cdots + U^{r-1}|\phi\rangle)$

Applying U to $|\Psi\rangle$, $U|\Psi_0\rangle = \frac{1}{\sqrt{r}}(U|\phi\rangle + U^2|\phi\rangle + \cdots + U^r|\phi\rangle)$

Considering that $U^r|\phi\rangle = |\phi\rangle$, then we get
$U|\Psi_0\rangle = \frac{1}{\sqrt{r}}(U|\phi\rangle + U^2|\phi\rangle + \cdots + |\phi\rangle) = |\Psi_0\rangle$

$\therefore$ $|\Psi_0\rangle$ is an eigenvector of U with assoc. eigenvalue 1.

Considering, $|\Psi_1\rangle = \frac{1}{\sqrt{r}}(|\phi\rangle + \exp\left[\frac{-2\pi i}{r}\right]U|\phi\rangle + |\phi\rangle +$
$\exp\left[\frac{-2\pi i}{r} \times 2\right]U^2|\phi\rangle + \cdots + \exp\left[\frac{-2\pi i}{r} \times (r-1)\right]U^{r-1}|\phi\rangle)$

i.e. adding an additional phase in each ot the terms.

Applying U to $|\Psi_1\rangle$, we can show that $|\Psi_1\rangle$ is also an eigenvector of U but with a different eigenvalue. $U|\Psi_1\rangle = \exp\left[\frac{-2\pi i}{r}\right]|\Psi_1\rangle$

Period Finding – generalization

Let's consider now,

$$|\Psi_s\rangle = \frac{1}{r}\left(|\phi\rangle + e^{\left[-\frac{2\pi i}{r}s\right]}U|\phi\rangle + e^{\left[-\frac{2\pi i}{r}2s\right]}U^2|\phi\rangle + \dots + e^{\left[-\frac{2\pi i}{r}(r-1)s\right]}U^{(r-1)}|\phi\rangle\right)$$

with $s \in \mathbb{Z}$ between 0 and $r - 1$

Period Finding – generalization

Let's consider now,

$$|\Psi_s\rangle = \frac{1}{r}\left(|\phi\rangle + e^{\left[-\frac{2\pi i}{r}s\right]}U|\phi\rangle + e^{\left[-\frac{2\pi i}{r}2s\right]}U^2|\phi\rangle + \cdots + e^{\left[-\frac{2\pi i}{r}(r-1)s\right]}U^{(r-1)}|\phi\rangle\right)$$

with $s \in \mathbb{Z}$ between 0 and $r-1$

$\Rightarrow |\Psi_s\rangle$ is an eigenvector of U with eigenvalue $e^{\left[\frac{2\pi i}{r}s\right]}$

Period Finding – generalization

Let's consider now,

$$|\Psi_s\rangle = \frac{1}{r}\left(|\phi\rangle + e^{\left[-\frac{2\pi i}{r}s\right]}U|\phi\rangle + e^{\left[-\frac{2\pi i}{r}2s\right]}U^2|\phi\rangle + \cdots + e^{\left[-\frac{2\pi i}{r}(r-1)s\right]}U^{(r-1)}|\phi\rangle\right)$$

with $s \in \mathbb{Z}$ between 0 and $r - 1$

$\Rightarrow |\Psi_s\rangle$ is an eigenvector of U with eigenvalue $e^{\left[\frac{2\pi i}{r}s\right]}$

QPE = Quantum Phase Estimation
computes the eigenvalue associated with an eigenvector

Period Finding – generalization

Let's consider now,

$$|\Psi_s\rangle = \frac{1}{r}\left(|\phi\rangle + e^{\left[-\frac{2\pi i}{r}s\right]}U|\phi\rangle + e^{\left[-\frac{2\pi i}{r}2s\right]}U^2|\phi\rangle + \cdots + e^{\left[-\frac{2\pi i}{r}(r-1)s\right]}U^{(r-1)}|\phi\rangle\right)$$

with $s \in \mathbb{Z}$ between 0 and $r-1$

$\Rightarrow |\Psi_s\rangle$ is an eigenvector of U with eigenvalue $e^{\left[\frac{2\pi i}{r}s\right]}$

QPE = Quantum Phase Estimation
computes the eigenvalue associated with an eigenvector

I.e. if we could obtain any of these states $|\Psi_s\rangle$ and by applying the quantum phase estimation (QPE) algorithm, we could then obtain $\frac{r}{s} \Rightarrow$ period!

Period Finding – generalization

Let's consider now,

$$|\Psi_s\rangle = \frac{1}{r}\left(|\phi\rangle + e^{\left[-\frac{2\pi i}{r}s\right]}U|\phi\rangle + e^{\left[-\frac{2\pi i}{r}2s\right]}U^2|\phi\rangle + \cdots + e^{\left[-\frac{2\pi i}{r}(r-1)s\right]}U^{(r-1)}|\phi\rangle\right)$$

with $s \in \mathbb{Z}$ between 0 and $r-1$

$\Rightarrow |\Psi_s\rangle$ is an eigenvector of U with eigenvalue $e^{\left[\frac{2\pi i}{r}s\right]}$

QPE = Quantum Phase Estimation
computes the eigenvalue associated with an eigenvector

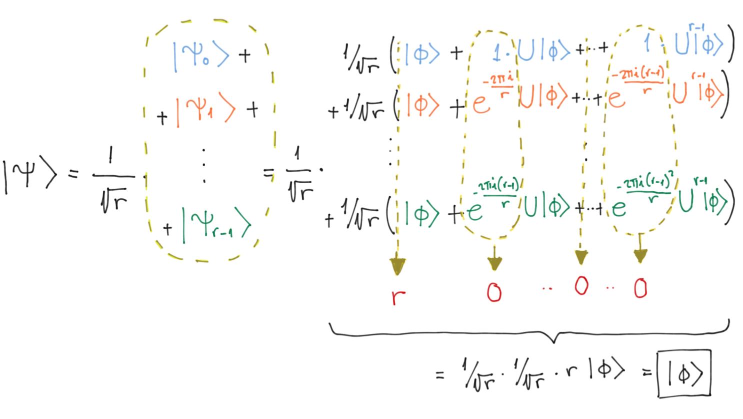
I.e. if we could obtain any of these states $|\Psi_s\rangle$ and by applying the quantum phase estimation (QPE) algorithm, we could then obtain $\frac{r}{s} \Rightarrow$ period!

However, there is a "chicken-egg" situation here...

So, “if you can not fix it” with enough states... then just add MORE states... i.e. create the superposition of all the $\boldsymbol{\Psi_s}$: $|\boldsymbol{\Psi}\rangle = \frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} |\boldsymbol{\Psi_s}\rangle$

So, “if you can not fix it” with enough states... then just add MORE states... i.e. create the superposition of all the $\boldsymbol{\Psi_s}$: $|\boldsymbol{\Psi}\rangle = \frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} |\boldsymbol{\Psi_s}\rangle = |\phi\rangle$

So, “if you can not fix it” with enough states... then just add MORE states... i.e. create the superposition of all the Ψ_s: $|\Psi\rangle = \frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} |\Psi_s\rangle = |\phi\rangle$

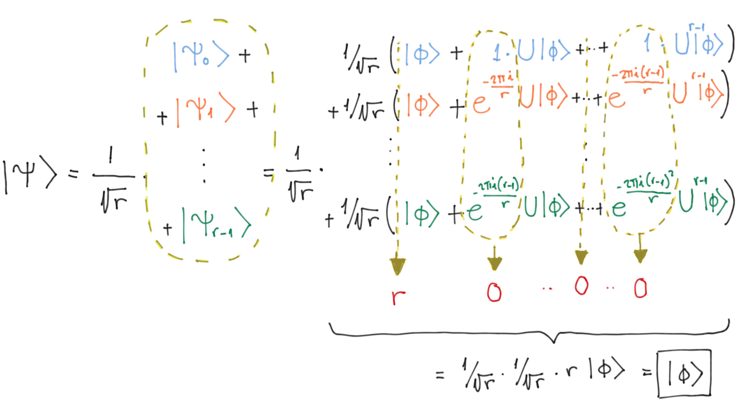


$$|\psi\rangle = \frac{1}{\sqrt{r}} \cdot \left(|\Psi_0\rangle + |\Psi_1\rangle + \cdots + |\Psi_{r-1}\rangle \right) = \frac{1}{\sqrt{r}} \cdot \Big[\tfrac{1}{\sqrt{r}} \left(|\phi\rangle + 1 \cdot U|\phi\rangle + \cdots + 1 \cdot U^{r-1}|\phi\rangle \right) + \tfrac{1}{\sqrt{r}} \left(|\phi\rangle + e^{\frac{-2\pi i}{r}} U|\phi\rangle + \cdots + e^{\frac{-2\pi i (r-1)}{r}} U^{r-1}|\phi\rangle \right) + \cdots + \tfrac{1}{\sqrt{r}} \left(|\phi\rangle + e^{\frac{-2\pi i (r-1)}{r}} U|\phi\rangle + \cdots + e^{\frac{-2\pi i (r-1)^2}{r}} U^{r-1}|\phi\rangle \right) \Big]$$

$$\underbrace{r \quad 0 \quad \cdots \quad 0 \quad \cdots \quad 0}$$

$$= \tfrac{1}{\sqrt{r}} \cdot \tfrac{1}{\sqrt{r}} \cdot r\, |\phi\rangle = \boxed{|\phi\rangle}$$

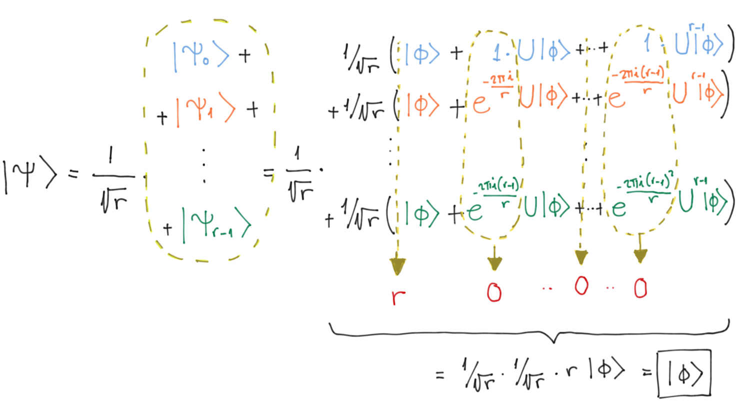
So, “if you can not fix it” with enough states... then just add MORE states... i.e. create the superposition of all the Ψ_s: $|\Psi\rangle = \frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} |\Psi_s\rangle = |\phi\rangle$



$|\Psi\rangle = \frac{1}{\sqrt{r}} \cdot$
$|\Psi_0\rangle +$
$+ |\Psi_1\rangle +$
$\vdots$
$+ |\Psi_{r-1}\rangle$
$= \frac{1}{\sqrt{r}} \cdot$
$1/\sqrt{r} \, (|\phi\rangle + 1 \cdot U|\phi\rangle + \dots + 1 \cdot U^{r-1}|\phi\rangle)$
$+ 1/\sqrt{r} \, (|\phi\rangle + e^{-\frac{2\pi i}{r}} U|\phi\rangle + \dots + e^{-\frac{2\pi i (r-1)}{r}} U^{r-1}|\phi\rangle)$
$\vdots$
$+ 1/\sqrt{r} \, (|\phi\rangle + e^{-\frac{2\pi i (r-1)}{r}} U|\phi\rangle + \dots + e^{-\frac{2\pi i (r-1)^2}{r}} U^{r-1}|\phi\rangle)$
r
0
$\cdots$
0
$\cdots$
0
$= 1/\sqrt{r} \cdot 1/\sqrt{r} \cdot r \, |\phi\rangle = \boxed{|\phi\rangle}$

$$\sum_{k=0}^{r-1} e^{\left[-\frac{2\pi i}{r} k\right]} = 0$$

So, "if you can not fix it" with enough states... then just add MORE states... i.e. create the superposition of all the $\mathbf{\Psi}_s$: $|\mathbf{\Psi}\rangle = \frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} |\mathbf{\Psi}_s\rangle = |\phi\rangle$



$$|\Psi\rangle = \frac{1}{\sqrt{r}} \cdot \left(|\Psi_0\rangle + {} + |\Psi_1\rangle + \vdots + |\Psi_{r-1}\rangle \right) = \frac{1}{\sqrt{r}} \cdot$$

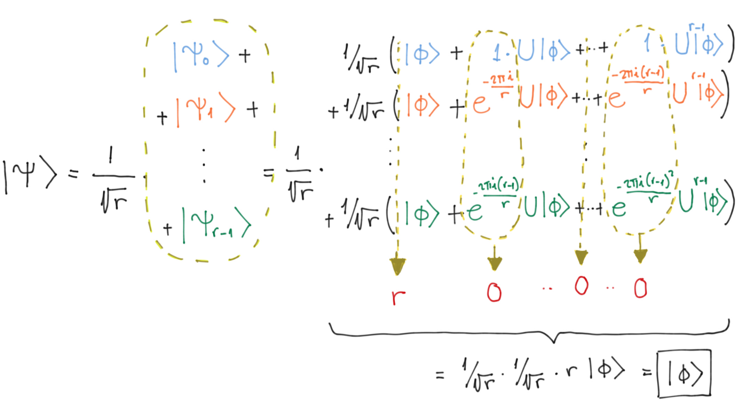
$$\underbrace{\begin{array}{l} 1/\sqrt{r}\left(|\phi\rangle + 1 \cdot U|\phi\rangle + \cdots + 1 \cdot U^{r-1}|\phi\rangle \right) \\ + 1/\sqrt{r}\left(|\phi\rangle + e^{-\frac{2\pi i}{r}} U|\phi\rangle + \cdots + e^{-\frac{2\pi i(r-1)}{r}} U^{r-1}|\phi\rangle \right) \\ \vdots \qquad\qquad\qquad \vdots \\ + 1/\sqrt{r}\left(|\phi\rangle + e^{-\frac{2\pi i(r-1)}{r}} U|\phi\rangle + \cdots + e^{-\frac{2\pi i(r-1)^2}{r}} U^{r-1}|\phi\rangle \right) \\ \downarrow \qquad \downarrow \qquad \downarrow \qquad \downarrow \\ r \qquad 0 \quad \cdot\cdot \quad 0 \quad \cdot\cdot \quad 0 \end{array}}$$

$$= 1/\sqrt{r} \cdot 1/\sqrt{r} \cdot r\,|\phi\rangle = \boxed{|\phi\rangle}$$

$$\sum_{k=0}^{r-1} e^{\left[-\frac{2\pi i}{r} k\right]} = 0$$

$\therefore$ $|\phi\rangle$ is just the *superposition* of all $|\mathbf{\Psi}_s\rangle$

So, “if you can not fix it” with enough states... then just add MORE states... i.e. create the superposition of all the $\boldsymbol{\Psi}_s$: $|\boldsymbol{\Psi}\rangle = \frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} |\boldsymbol{\Psi}_s\rangle = |\phi\rangle$

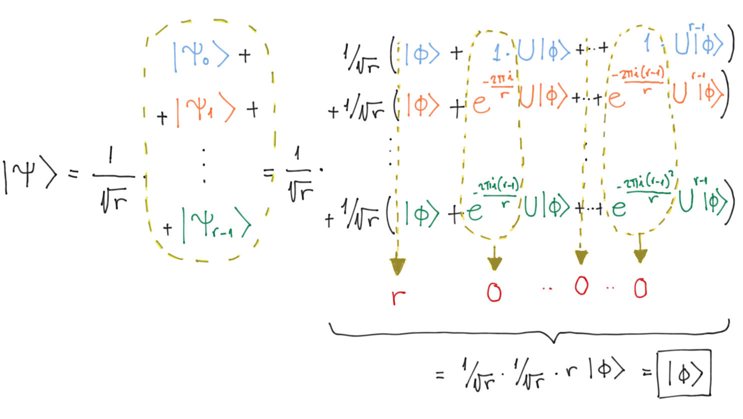

$|\Psi\rangle = \frac{1}{\sqrt{r}} \cdot \left(|\Psi_0\rangle + |\Psi_1\rangle + \cdots + |\Psi_{r-1}\rangle \right) = \frac{1}{\sqrt{r}} \cdot$
$1/\sqrt{r} \left(|\phi\rangle + 1 \cdot U|\phi\rangle + \cdots + 1 \cdot U^{r-1}|\phi\rangle \right)$
$+ 1/\sqrt{r} \left(|\phi\rangle + e^{-\frac{2\pi i}{r}} U|\phi\rangle + \cdots + e^{-\frac{2\pi i (r-1)}{r}} U^{r-1}|\phi\rangle \right)$
$\vdots$
$+ 1/\sqrt{r} \left(|\phi\rangle + e^{-\frac{2\pi i (r-1)}{r}} U|\phi\rangle + \cdots + e^{-\frac{2\pi i (r-1)^2}{r}} U^{r-1}|\phi\rangle \right)$
r
0 .. 0 .. 0
$= 1/\sqrt{r} \cdot 1/\sqrt{r} \cdot r\,|\phi\rangle = \boxed{|\phi\rangle}$

$$\sum_{k=0}^{r-1} e^{\left[-\frac{2\pi i}{r} k\right]} = 0$$

$\therefore$ $|\phi\rangle$ is just the *superposition* of all $|\boldsymbol{\Psi}_s\rangle$

Applying the QPE alg to the operator U $|\phi\rangle \rightsquigarrow$ *superposition* of all $\frac{r}{s}, s \in \{0, r-1\}$

So, "if you can not fix it" with enough states... then just add MORE states... i.e. create the superposition of all the $\boldsymbol{\Psi}_s$: $|\boldsymbol{\Psi}\rangle = \frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} |\boldsymbol{\Psi}_s\rangle = |\phi\rangle$



$|\Psi\rangle = \frac{1}{\sqrt{r}} \cdot \left(|\Psi_0\rangle + |\Psi_1\rangle + \cdots + |\Psi_{r-1}\rangle \right) = \frac{1}{\sqrt{r}} \cdot$
$1/\sqrt{r} \left(|\phi\rangle + 1 \cdot U|\phi\rangle + \cdots + 1 \cdot U^{r-1}|\phi\rangle \right)$
$+ 1/\sqrt{r} \left(|\phi\rangle + e^{-\frac{2\pi i}{r}} U|\phi\rangle + \cdots + e^{-\frac{2\pi i(r-1)}{r}} U^{r-1}|\phi\rangle \right)$
$+ 1/\sqrt{r} \left(|\phi\rangle + e^{-\frac{2\pi i(r-1)}{r}} U|\phi\rangle + \cdots + e^{-\frac{2\pi i(r-1)^2}{r}} U^{r-1}|\phi\rangle \right)$
r
0
0
0
$= 1/\sqrt{r} \cdot 1/\sqrt{r} \cdot r\,|\phi\rangle = \boxed{|\phi\rangle}$

$$\sum_{k=0}^{r-1} e^{\left[-\frac{2\pi i}{r}k\right]} = 0$$

$\therefore$ $|\phi\rangle$ is just the *superposition* of all $|\boldsymbol{\Psi}_s\rangle$

Applying the QPE alg to the operator U $|\phi\rangle \rightsquigarrow$ *superposition* of all $\frac{r}{s}$, $s \in \{0, r-1\}$

Measurements ⟶ decimal numbers ⟶ fraction $\frac{r}{s} \Rightarrow r$

Period Finding - Recap

Period Finding - Recap

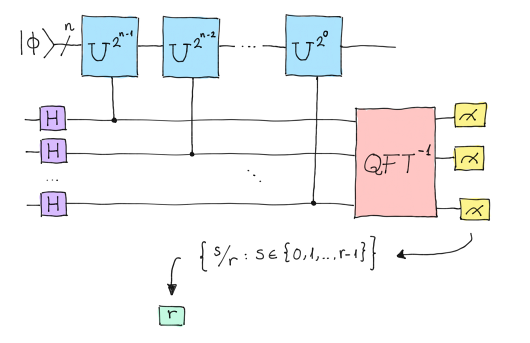
OBJECTIVE: Obtain the period from decimals

Period Finding - Recap

OBJECTIVE: Obtain the period from decimals

Given an operator U and a state $|\phi\rangle$, find an integer r such that: $U^r|\phi\rangle = |\phi\rangle$

Quantum Phase Estimation (QPE) on this state, will receive a superposition of states with amplitudes containing the different values of $\frac{s}{r}$, and by performing a series of measurements, will recover the value of r.



$\Rightarrow$ `qml.QuantumPhaseEstimation`

https://pennylane.readthedocs.io/en/stable/code/api/pennylane.QuantumPhaseEstimation.html

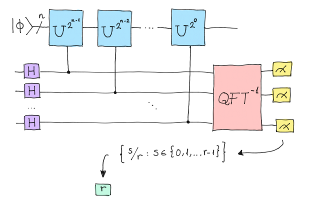
After obtaining the potential r values, one would take the **largest** of all possible fractions or, more efficiently, calculate the **least common multiple** of all of them.

QPE – Hands-on

Implement the QPE circuit for the state $|0001\rangle$

Useful functions:

`qml.QuantumPhaseEstimation`, `get_unitary_matrix`, `get_phase`



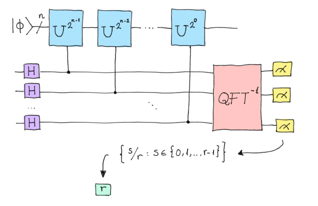
(Credit: Xanadu)

QPE – Hands-on

Implement the QPE circuit for the state $|0001\rangle$

Useful functions:

`qml.QuantumPhaseEstimation`, `get_unitary_matrix`, `get_phase`



(Credit: Xanadu)

```python
def U():
    qml.SWAP(wires=[2,3])
    qml.SWAP(wires=[1,2])
    qml.SWAP(wires=[0,1])
    for i in range(4):
        qml.PauliX(wires=i)

matrix = get_unitary_matrix(U, wire_order=range(4))()

n_target_wires = 4
target_wires = range(n_target_wires)
n_estimation_wires = 3
estimation_wires = range(4, 4 + n_estimation_wires)


```

```python
@qml.qnode(dev)
def circuit(matrix):
    """Return a sample after taking a shot at the estimation wires.

    Args:
        matrix (array[complex]): matrix representation of U.

    Returns:
        array[float]: a sample after taking a shot at the estimation wires.
    """
```

```python
@qml.qnode(dev)
def circuit(matrix):
    """Return a sample after taking a shot at the estimation wires.

    Args:
        matrix (array[complex]): matrix representation of U.

    Returns:
        array[float]: a sample after taking a shot at the estimation wires.
    """
```

```python
    # CREATE THE INITIAL STATE |0001> ON TARGET WIRES
    # penylane initializes states on |0>
    qml.PauliX(wires=3)
    # USE THE SUBROUTINE QUANTUM PHASE ESTIMATION
    qml.QuantumPhaseEstimation(matrix,target_wires,estimation_wires)

    return qml.sample(wires=estimation_wires)
```

```python
@qml.qnode(dev)
def circuit(matrix):
    """Return a sample after taking a shot at the estimation wires.

    Args:
        matrix (array[complex]): matrix representation of U.

    Returns:
        array[float]: a sample after taking a shot at the estimation wires.
    """
```

```python
    # CREATE THE INITIAL STATE |0001> ON TARGET WIRES
    # penylane initializes states on |0>
    qml.PauliX(wires=3)
    # USE THE SUBROUTINE QUANTUM PHASE ESTIMATION
    qml.QuantumPhaseEstimation(matrix,target_wires,estimation_wires)

    return qml.sample(wires=estimation_wires)
```

```python
def get_phase(matrix):
    binary = "".join([str(b) for b in circuit(matrix)])
    return int(binary, 2) / 2 ** n_estimation_wires
```

```python
for i in range(5):
    print(circuit(matrix))
    print(f"shot␣{i+1},␣phase:",get_phase(matrix))
```

```
[1 1 0]
shot 1, phase: 0.25
[0 1 0]
shot 2, phase: 0.5
[1 0 0]
shot 3, phase: 0.25
[1 0 0]
shot 4, phase: 0.25
[1 1 0]
shot 5, phase: 0.0
```

Hands-on: Period Finding i

```python
def U():
    qml.SWAP(wires=[2,3])
    qml.SWAP(wires=[1,2])
    qml.SWAP(wires=[0,1])
    for i in range(4):
        qml.PauliX(wires=i)

matrix = get_unitary_matrix(U, wire_order=range(4))()

target_wires = range(4)
n_estimation_wires = 3
estimation_wires = range(4, 4 + n_estimation_wires)
```

Hands-on: Period Finding ii

```python
def get_period(matrix):
    """Return the period of the state using the already-defined
    get_phase function.

    Args:
        matrix (array[complex]): matrix associated with the operator U

    Returns:
        int: Obtained period of the state.
    """

    shots = 10

    ##################
    r = 0
    for i in range(shots):
        phase = Fraction(get_phase(matrix)).denominator
        if (phase > r):
            r = phase
        print(f"shot␣{i+1},␣phase:",phase, Fraction(phase))

    return(r)
```

Hands-on: Period Finding iii

```python
print(get_period(matrix))
```

```
shot 1, phase: 4 4
shot 2, phase: 4 4
shot 3, phase: 2 2
shot 4, phase: 4 4
shot 5, phase: 4 4
shot 6, phase: 2 2
shot 7, phase: 1 1
shot 8, phase: 4 4
shot 9, phase: 1 1
shot 10, phase: 4 4
4
```

Shor's Algorithm – relationship between period-finding & nontrivial-square-root

Nontrivial square root: $x^2 \equiv 1 \pmod{N}$, or more generally, $x^r \equiv 1 \pmod{N}$ with r an even number.

Shor's Algorithm – relationship between period-finding & nontrivial-square-root

Nontrivial square root: $x^2 \equiv 1 \pmod{N}$, or more generally, $x^r \equiv 1 \pmod{N}$ with r an even number.

Let's define the following function, $f_{N,a}(m) = a^m \pmod{m}$

Shor’s Algorithm – relationship between period-finding & nontrivial-square-root

Nontrivial square root: $x^2 \equiv 1 \pmod{N}$, or more generally, $x^r \equiv 1 \pmod{N}$ with r an even number.

Let’s define the following function, $f_{N,a}(m) = a^m \pmod{m}$

We want to know which m satisfies $f_{N,a}(m) = 1$, i.e. $a^m \equiv 1 \pmod{m}$, because if m is even $\Rightarrow$ nontrivial square root.

Shor's Algorithm – relationship between period-finding & nontrivial-square-root

Nontrivial square root: $x^2 \equiv 1 \pmod{N}$, or more generally, $x^r \equiv 1 \pmod{N}$ with r an even number.

Let's define the following function, $f_{N,a}(m) = a^m \pmod{m}$

We want to know which m satisfies $f_{N,a}(m) = 1$, i.e. $a^m \equiv 1 \pmod{m}$, because if m is even $\Rightarrow$ nontrivial square root.

Also, by the defn. f is a periodic function with period m

$\Rightarrow$ finding the period $\approx$ finding the nontrivial square root.

Shor's alg. proposes to define an operator $U_{N,a}$ such that $\boxed{U^m_{N,a}\,|1\rangle = |f_{N,a}(m)\rangle}$

$\Rightarrow$ finding an r such that $f^r_{N,a} = 1$ would be equivalent to finding the minimum r such that $U^r_{N,a}\,|1\rangle \equiv |1\rangle \Leftarrow$ period finding alg.

Full Algorithm

1. Select a random number a between 2 and $N-2$ (Classical).
2. Check that a and N are not coprime numbers (Classical).
3. Construct the operator $U_{N,a}$ and calculate its period r (Period Finding: Quantum).
4. If r is odd, go back to step 1.
5. Compute the nontrivial square root x as $a^{r/2}$ (Classical).
6. Compute the factors of N as $GCD(x-1,)$ and $GCD(x+1,N)$ (Classical).

Shor's Algorithm – Some Observations

- $U_{N,a}$ can only be implemented in a QC if a and N are coprime, oherwise there could be 2 $\neq$ elements $|x\rangle$ and $|y\rangle$ such that $U|x\rangle = U|y\rangle \Rightarrow U$ would not be invertible
- Shor's algorithm only uses quantum computation to find the period of the function
- Currently, quantum computers have proven to be much more efficient in a reduced set of algorithms, among which is the finding period of this type of functions
- Shor's insight was to translate the factorization of prime numbers to period finding one

Shor's Algorithm – Hands-on i

Implement the folowing helper functions:

- `is_coprime` receives two integers and returns True if they are *coprime* or False otherwise.
- `is_odd` given an integer returns True if it is *odd* and False otherwise.
- `is_not_one` given x and N, determine if $x \not\equiv \pm 1 \pmod{N}$. Return False if they are equal.

Obs:

Coprime numbers are those numbers that have only one common factor, namely 1. That means a pair of numbers are said to be co prime when they have their highest common factor as 1.

Shor's Algorithm – Hands-on ii

```python
def is_coprime(a, N):
    """Determine if two numbers are coprime.

    Args:
        a (int): First number to check if is coprime with the other.
        N (int): Second number to check if is coprime with the other.

    Returns:
        bool: True if they are coprime numbers, False otherwise.
    """

    if np.gcd(a,N) == 1:
        return True
    else:
        return False
```

Shor's Algorithm – Hands-on iii

```python
def is_odd(r):
    """Determine if a number is odd.

    Args:
        r (int): Integer to check if is an odd number.

    Returns:
        bool: True if it is odd, False otherwise.
    """

    if (r % 2 == 0):
        return False
    else:
        return True
```

Shor's Algorithm – Hands-on iv

```python
def is_not_one(x, N):
    """Determine if x is not +- 1 modulo N.

    Args:
        N (int): Modulus of the equivalence.
        x (int): Integer to check if it is different from +-1 modulo N.

    Returns:
        bool: True if it is different, False otherwise.
    """

    if ((x-1)%N==0) or ((x+1)%N==0):
        return False
    else:
        return True
```

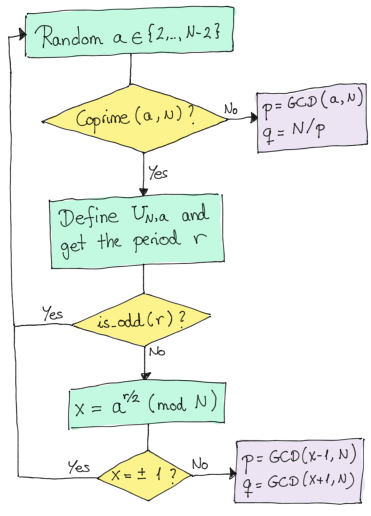
Shor’s Algorithm – Hands-on v

```python
print("3 and 12 are coprime numbers: ", is_coprime(3,12))
print("5 is odd: ", is_odd(5))
print("4 is not one mod 5: ",is_not_one(4,5))


# 3 and 12 are coprime numbers: False
# 5 is odd: True
# 4 is not one mod 5: False
```

Hands-on: Shor's Algorithm complete implementation

Implement Shor's algorithm as shown in the flow diagram:



The RSA System

Cryptography - Some Basic Definitions i

Encryption The process of converting the message from its plaintext to ciphertext

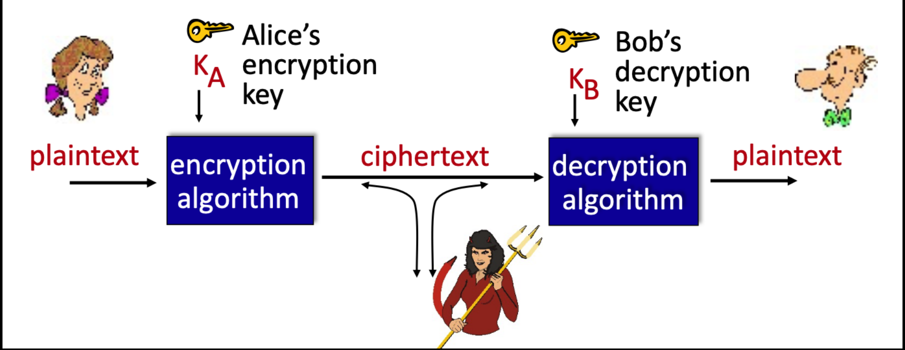
Plaintext The message in its natural format has not been turned into a secret.

Ciphertext The altered form of a plaintext message, so as to be unreadable for anyone except the intended recipients. Something that has been turned into a secret.

Hash function Function that accepts an input message of any length and generates, through a one-way operation, a fixed-length output called a message digest or hash.

Source: `https://www.isc2.org/Certifications/CISSP/CISSP-Student-Glossary`

Cryptography - Some Basic Definitions ii



m plaintext message

$K_A(m)$ ciphertext, encrypted with key K_A

$m = K_B(K_A(m))$ ciphertext decrypted

RSA

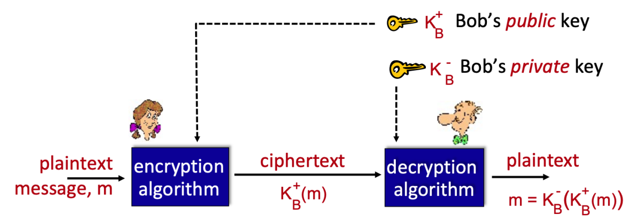
- RSA $=$ Rivest, Shamir, Adelson algorithm
- the rough estimate is that QC will need $\approx \mathbf{10^3}$ qubits in order to break current cybersecturiyt schema

RSA

- RSA = Rivest, Shamir, Adelson algorithm
- the rough estimate is that QC will need $\approx 10^3$ qubits in order to break current cybersecturiyt schema

Public Key Cryptographic System

- aka Asymmetric Key Cryptography
- sender and receiver do not share secret key
- public encryption key is known to all
- private decryption key is known only to receiver



Public Keys Systems

1. Bob will create a method for encoding messages and another method for decoding messages.
2. Bob will send Alice the instructions to encode any message. These instructions are what we formally denote as the *public key*, as when this key is sent over the communication channel, it can be intercepted by Trudy.
3. Alice encrypts her message for Bob using the public key and sends it to him. In the same way, Trudy can intercept this message.
4. Bob is the only one who knows the decoding method, which we will call *private key*. Therefore, even if someone else knows the encrypted message and the public key, they will not be able to decode it.

A tell of symmetric keys

It may seem that if we know how a message is encoded, we can deduce the decoding method, but this process may be very expensive.

Bob sends a 128-integers message

Suppose Bob knows a matrix M of size 128 and also knows its inverse M^{-1}

Bob will send Alice the matrix M, so Alice can use it to encode her message

$\Rightarrow$ Alice's encoded message $\vec{c} = M\vec{m}$

Bob will recover the original message $\vec{m}$ by applying M^{-1}, this matrix has not been made public at any time:

$$M^{-1}\vec{c} = M^{-1}M\vec{m} = \vec{m}$$

Someone who knows M (the public key) could recover M^{-1} (the private key), but the process of calculating the inverse of a matrix is an expensive process.

Symmetric vs Asymmetric Keys
RSA uses methods for which calculation of the private key is totally intractable for large values (with a traditional computer).

RSA Algorithm

- Choose two prime numbers p and q
- Define $N = pq$
- Define $\theta = (p-1)(q-1)$
- Choose an e that is coprime with θ
- Compute d as the inverse of $e \pmod{\theta}$

RSA Algorithm

- Choose two prime numbers p and q
- Define $N = pq$
- Define $\theta = (p-1)(q-1)$
- Choose an e that is coprime with θ
- Compute d as the inverse of $e \pmod{\theta}$

The public key will be the pair (e, N) and an integer message m will be encrypted as:

$$\boxed{c = m^e \pmod{N}}$$

The private key, known only by Bob, is defined by the pair (d, N). The process to obtain the original message is as follows:

$$\boxed{m = c^d \pmod{N}}$$

RSA Algorithm

- Choose two prime numbers p and q
- Define $N = pq$
- Define $\theta = (p-1)(q-1)$
- Choose an e that is coprime with θ
- Compute d as the inverse of $e \pmod{\theta}$

The public key will be the pair (e, N) and an integer message m will be encrypted as:

$$\boxed{c = m^e \pmod{N}}$$

The private key, known only by Bob, is defined by the pair (d, N). The process to obtain the original message is as follows:

$$\boxed{m = c^d \pmod{N}}$$

Example,
$p = 5, q = 13 \Rightarrow N = 5 \cdot 13 = 65$
and $\theta = (5-1)(13-1) = 48$

RSA Algorithm

- Choose two prime numbers p and q
- Define $N = pq$
- Define $\theta = (p-1)(q-1)$
- Choose an e that is coprime with θ
- Compute d as the inverse of e (mod θ)

The public key will be the pair (e, N) and an integer message m will be encrypted as:

$$\boxed{c = m^e \pmod N}$$

The private key, known only by Bob, is defined by the pair (d, N). The process to obtain the original message is as follows:

$$\boxed{m = c^d \pmod N}$$

Example,
$p = 5, q = 13 \Rightarrow N = 5 \cdot 13 = 65$
and $\theta = (5-1)(13-1) = 48$
Choosing $e = 5$, as it is coprime with θ, and calculate its inverse $d = 29$

RSA Algorithm

- Choose two prime numbers p and q
- Define $N = pq$
- Define $\theta = (p-1)(q-1)$
- Choose an e that is coprime with θ
- Compute d as the inverse of $e \pmod{\theta}$

The public key will be the pair (e, N) and an integer message m will be encrypted as:

$$\boxed{c = m^e \pmod{N}}$$

The private key, known only by Bob, is defined by the pair (d, N). The process to obtain the original message is as follows:

$$\boxed{m = c^d \pmod{N}}$$

Example,
$p = 5, q = 13 \Rightarrow N = 5 \cdot 13 = 65$
and $\theta = (5-1)(13-1) = 48$
Choosing $e = 5$, as it is coprime with θ, and calculate its inverse $d = 29$
For sending the number "6", it should be encoded using the $c = m^e \pmod{N}$ eqn. $c = 6^5 \pmod{65} = 41 \pmod{65}$

RSA Algorithm

- Choose two prime numbers p and q
- Define $N = pq$
- Define $\theta = (p-1)(q-1)$
- Choose an e that is coprime with θ
- Compute d as the inverse of $e \pmod{\theta}$

The public key will be the pair (e, N) and an integer message m will be encrypted as:

$$\boxed{c = m^e \pmod{N}}$$

The private key, known only by Bob, is defined by the pair (d, N). The process to obtain the original message is as follows:

$$\boxed{m = c^d \pmod{N}}$$

Example,
$p = 5, q = 13 \Rightarrow N = 5 \cdot 13 = 65$
and $\theta = (5-1)(13-1) = 48$
Choosing $e = 5$, as it is coprime with θ, and calculate its inverse $d = 29$
For sending the number "6", it should be encoded using the $c = m^e \pmod{N}$ eqn. $c = 6^5 \pmod{65} = 41 \pmod{65}$
To obtain the original msg, we can verify that $41^{29} \pmod{65} = 6$

Breaking RSA

Qn: can we recover the private key from the public one?

(e, N) is given as it is teh publci key, and we would like to know (d, N)

Breaking RSA

Qn: can we recover the private key from the public one?
(e, N) is given as it is teh publci key, and we would like to know (d, N)
d is unknown, that is, the inverse of $e \pmod{\theta}$

Breaking RSA

Qn: can we recover the private key from the public one?

(e, N) is given as it is teh publci key, and we would like to know (d, N)

d is unknown, that is, the inverse of $e \pmod{\theta}$

To obtain that inverse, we need to determine

$\theta = (p-1)(q-1)$

Breaking RSA

Qn: can we recover the private key from the public one?

(e, N) is given as it is teh publci key, and we would like to know (d, N)

d is unknown, that is, the inverse of $e \pmod{\theta}$

To obtain that inverse, we need to determine

$\theta = (p-1)(q-1)$

$\Rightarrow p, q$

Breaking RSA

Qn: can we recover the private key from the public one?

(e, N) is given as it is teh publci key, and we would like to know (d, N)

d is unknown, that is, the inverse of $e \pmod{\theta}$

To obtain that inverse, we need to determine

$\theta = (p-1)(q-1)$

$\Rightarrow p, q$

decompsing N into its prime factors $\rightsquigarrow p$ and q

Breaking RSA

Qn: can we recover the private key from the public one?

(e, N) is given as it is teh publci key, and we would like to know (d, N)

d is unknown, that is, the inverse of $e \pmod{\theta}$

To obtain that inverse, we need to determine

$\theta = (p-1)(q-1)$

$\Rightarrow p, q$

decompsing N into its prime factors $\rightsquigarrow p$ and q

Suppose we have intercepted that the public key is the pair (5,35) and the encrypted message 3.

Breaking RSA

Qn: can we recover the private key from the public one?

(e, N) is given as it is teh publci key, and we would like to know (d, N)

d is unknown, that is, the inverse of $e \pmod{\theta}$

To obtain that inverse, we need to determine

$\theta = (p-1)(q-1)$

$\Rightarrow p, q$

decompsing N into its prime factors $\rightsquigarrow$ p and q

Suppose we have intercepted that the public key is the pair (5,35) and the encrypted message 3.

To decode the message we first need to know the prime factors of 35

Breaking RSA

Qn: can we recover the private key from the public one?

(e, N) is given as it is teh publci key, and we would like to know (d, N)

d is unknown, that is, the inverse of $e \pmod{\theta}$

To obtain that inverse, we need to determine

$\theta = (p-1)(q-1)$

$\Rightarrow p, q$

decompsing N into its prime factors $\rightsquigarrow p$ and q

Suppose we have intercepted that the public key is the pair (5,35) and the encrypted message 3.

To decode the message we first need to know the prime factors of 35 $\longrightarrow p = 5$ and $q = 7$

Breaking RSA

Qn: can we recover the private key from the public one?

(e, N) is given as it is teh publci key, and we would like to know (d, N)

d is unknown, that is, the inverse of $e \pmod{\theta}$

To obtain that inverse, we need to determine

$\theta = (p-1)(q-1)$

$\Rightarrow p, q$

decompsing N into its prime factors $\rightsquigarrow p$ and q

Suppose we have intercepted that the public key is the pair (5,35) and the encrypted message 3.

To decode the message we first need to know the prime factors of 35 $\longrightarrow p = 5$ and $q = 7 \Rightarrow \theta = (5-1)(7-1) = 24$

Breaking RSA

Qn: can we recover the private key from the public one?

(e, N) is given as it is teh publci key, and we would like to know (d, N)

d is unknown, that is, the inverse of $e \pmod{\theta}$

To obtain that inverse, we need to determine

$\theta = (p - 1)(q - 1)$

$\Rightarrow p, q$

decompsing N into its prime factors $\rightsquigarrow p$ and q

Suppose we have intercepted that the public key is the pair (5,35) and the encrypted message 3.

To decode the message we first need to know the prime factors of 35 $\longrightarrow p = 5$ and $q = 7 \Rightarrow \theta = (5 - 1)(7 - 1) = 24$

Calculating the inverse of 5 (mod 24) $\longrightarrow$ $d = 5$

Breaking RSA

Qn: can we recover the private key from the public one?

(e, N) is given as it is teh publci key, and we would like to know (d, N)

d is unknown, that is, the inverse of $e \pmod{\theta}$

To obtain that inverse, we need to determine $\theta = (p-1)(q-1)$

$\Rightarrow p, q$

decompsing N into its prime factors $\rightsquigarrow p$ and q

Suppose we have intercepted that the public key is the pair (5,35) and the encrypted message 3.

To decode the message we first need to know the prime factors of 35 $\rightarrow p = 5$ and $q = 7 \Rightarrow \theta = (5-1)(7-1) = 24$

Calculating the inverse of 5 (mod 24) $\rightarrow$ $d = 5$

so, we can recover the message $m = c^d \pmod{N} = 3^5$ (mod 35) $= \boxed{33}$

Breaking RSA

Qn: can we recover the private key from the public one?

(e, N) is given as it is teh publci key, and we would like to know (d, N)

d is unknown, that is, the inverse of $e \pmod{\theta}$

To obtain that inverse, we need to determine

$\theta = (p-1)(q-1)$

$\Rightarrow p, q$

decompsing N into its prime factors $\rightsquigarrow p$ and q

Suppose we have intercepted that the public key is the pair (5,35) and the encrypted message 3.

To decode the message we first need to know the prime factors of 35 → $p = 5$ and $q = 7 \Rightarrow \theta = (5-1)(7-1) = 24$

Calculating the inverse of 5 (mod 24) → $d = 5$

so, we can recover the message $m = c^d \pmod{N} = 3^5$ (mod 35) $= \boxed{33}$

Decoding was possible because the number was very small and doable to factorize

Currently, RSA uses primes of the order of 600 digits, which makes factoring in the traditional way an intractable process.

RSA in practice

- exponentiation in RSA is computationally intensive
- DES is at least 100 times faster than RSA
- use public key crypto to establish secure connection, then establish second key – symmetric session key – for encrypting data
- Bob and Alice use RSA to exchange a symmetric session key KS
- once both have KS, they use symmetric key cryptography

RSA – Hands-on

Create a function that, given two primes p and q returns valid values of e, d and N.

RSA – Hands-on

Create a function that, given two primes p and q returns valid values of e, d and N.

```python
def create_keys(p, q):
    """Returns the characteristic e, d and N values of RSA

    Args:
        p (int): First prime number of the algorithm.
        q (int): Second prime number of the algorithm.

    Returns:
        (int, int, int): a tuple consisting of the ’e’ value of the RSA codification.
             ’d’ value of the RSA codification.
            and ’N’, the product of p and q.
    """
```

RSA – Hands-on

Create a function that, given two primes p and q returns valid values of e, d and N.

```
def create_keys(p, q):
    """Returns the characteristic e, d and N values of RSA

    Args:
        p (int): First prime number of the algorithm.
        q (int): Second prime number of the algorithm.

    Returns:
        (int, int, int): a tuple consisting of the ’e’ value of the RSA codification. ’d’ value of the RSA codification.
            and ’N’, the product of p and q.
    """
```

```
    # compute theta by defn
    theta = (p-1)*(q-1)

    done = False
    while (not done):
        e = np.random.randint(0,p)
  # check that e  and theta are coprimes
        if (np.gcd(e,theta)==1):
            done = True
    d = pow(e,-1,theta)
    N = p * q

    return (e,d,N)
```

```
print(create_keys(3,53))

# (1, 1, 159)
```

Create a function that given $\boldsymbol{d, N}$ and a series of integers $[\boldsymbol{c0, c1, ...}]$, is able to retrieve the original message. Pass each of the received numbers to 'ascii' to read the hidden message.

Create a function that given d, N and a series of integers $[c0, c1, ...]$, is able to retrieve the original message. Pass each of the received numbers to 'ascii' to read the hidden message.

```python
def decode(d,N, code):
    """Decode an encrypted message

    Args:
        d (int): Value of the RSA codification.
        N (int): Product of p and q.
        code list[int]: List of values to be decoded.

    Returns:
        string: Decoded message. (One character per list item)
    """
```

Create a function that given d, N and a series of integers $[c0, c1, ...]$, is able to retrieve the original message. Pass each of the received numbers to 'ascii' to read the hidden message.

```python
def decode(d,N, code):
    """Decode an encrypted message

    Args:
        d (int): Value of the RSA codification.
        N (int): Product of p and q.
        code list[int]: List of values to be decoded.

    Returns:
        string: Decoded message. (One character per list item)
    """
```

```python
    message = ""

    # m =c^d (mod N)
    for c in code:
        message = message + chr(c**d % N)

    return message
```

Post-Quantum Cryptography

- By 2016, NIST started the search for possible *Post Quantum Encryption* candidates:
  `https://csrc.nist.gov/Projects/post-quantum-cryptography`
- Main idea would be to have a modular implementation of cryptographical features
- But, one should also consider encrypted storage and silos.

Some Final Remarks

A lot has been left out...

- A lot of math and physics has not been covered/discussed/presented
- Some concrete algorithms were not covered at full:

  Quantum Phase Estimation

  Period Finding

  Linear systems of equations

  Hamiltonean Dynamics

  QML

Some final thoughts... i

- QC is (at the moment) a lot (like a lot) of linear algebra, complex numbers, probability theory (on steroids), ...
- QCers do NOT work by trying multiple parallel solutions at once
  Take a look at this “comic strip” by Scott Aaronson & Zach Weinersmith
  `https://www.smbc-comics.com/comic/the-talk-3`
- QCers work by harvesting the most involved and “weird” concepts of quantum mechanics: superposition, entanglement, interference

Some final thoughts... ii

- The way to program QCers (right now) is via *quantum circuits* $\leftrightsquigarrow$ *gates* or operators

  Some may say, that programming a QC is finding the right circuit, ie. a combination of gates/ops. to generate the proper interference pattern in your quantum states (think about the double-slit experiment, this is what is going on but at a much larger scale)

  Also, the current APIs are more like “quantum circuitry assembly” language
- Be aware and careful (critical?) of the “hype”, e.g. quantum supremacy, quantum advantage, ...
- QC won't solve all type of problems, but a set of problems that traditional or classical computers have a hard time solving, e.g. high ”complexity” (CS) ones, or “weird” ones