

# **CPEN 400Q Lecture 12**

## **Order finding and Shor's algorithm**

Friday 17 February 2023

# Announcements

- First questions of assignment 2 available very soon (final one still in testing)
- Project group and paper selection due **today** (use Piazza to find teammates)

## Last time

We implemented quantum phase estimation.

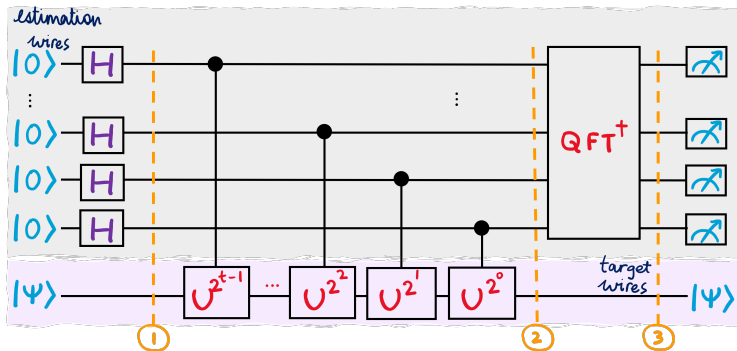
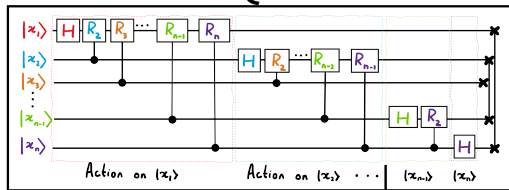
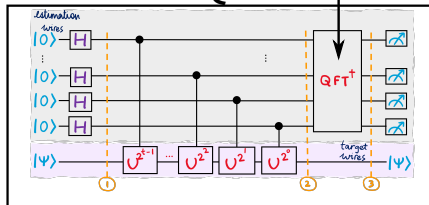


Image credit: Xanadu Quantum Codebook node P.2

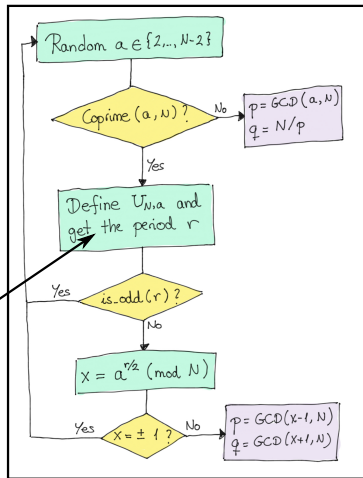
## 1. QFT



## 2. QPE



## 3. Shor



- Use QPE to implement the order finding algorithm
- Implement both classical and quantum components of Shor's algorithm to factor numbers
- Describe the steps involved in the RSA cryptosystem, and identify the vulnerability to quantum computers

# Order finding on a quantum computer

Suppose we have a function

over the integers modulo  $N$ .

If there exists  $r \in \mathbb{Z}$  s.t.

$f(x)$  is periodic with period  $r$ .

# Order finding on a quantum computer

Suppose

The *order* of  $a$  is the smallest  $m$  such that

Note that this is also the period:

# Order finding on a quantum computer

More formally, define

Define a unitary operation that performs

If  $m$  is the order of  $a$ , and we apply  $U_{N,a}$   $m$  times,

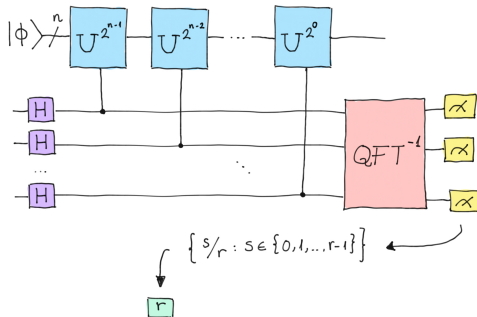
So  $m$  is also the order of  $U_{N,a}$ ! We can find it efficiently using a quantum computer.



# Order finding on a quantum computer

Let  $U$  be an operator and  $|\phi\rangle$  any state. How do we find the minimum  $r$  such that

QPE does the trick if we set things up in a clever way:



# Order finding on a quantum computer

Consider the state

If we apply  $U$  to this:

## Order finding on a quantum computer

Now consider the state

If we apply  $U$  to this:

## Order finding on a quantum computer

This generalizes to  $|\Psi_s\rangle$

It has eigenvalue

Idea: if we can create *any* one of these  $|\Psi_s\rangle$ , we could run QPE and get an estimate for  $s/r$ , and then recover  $r$ .

## Order finding on a quantum computer

Problem: to construct any  $|\Psi_s\rangle$ , we would need to know  $r$  in advance!

Solution: construct the uniform superposition of all of them.

But what does this equal?

# Order finding on a quantum computer

The superposition of all  $|\psi_s\rangle$  is just our original state  $|\phi\rangle$ !

$$\begin{aligned}
 |\psi\rangle &= \frac{1}{\sqrt{r}} \left( |\psi_0\rangle + |\psi_1\rangle + \dots + |\psi_{r-1}\rangle \right) \\
 &= \frac{1}{\sqrt{r}} \cdot \left( \frac{1}{\sqrt{r}} (|\phi\rangle + e^{\frac{-2\pi i}{r}} U|\phi\rangle + \dots + e^{\frac{-2\pi i(r-1)}{r}} U^{r-1}|\phi\rangle) \right. \\
 &\quad \left. + \frac{1}{\sqrt{r}} (|\phi\rangle + e^{\frac{-2\pi i}{r}} U|\phi\rangle + \dots + e^{\frac{-2\pi i(r-1)}{r}} U^{r-1}|\phi\rangle) \right. \\
 &\quad \left. + \dots + \frac{1}{\sqrt{r}} (|\phi\rangle + e^{\frac{-2\pi i(r-1)}{r}} U|\phi\rangle + \dots + e^{\frac{-2\pi i(r-1)}{r}} U^{r-1}|\phi\rangle) \right) \\
 &\quad \underbrace{\qquad\qquad\qquad}_{\substack{r \\ = \frac{1}{\sqrt{r}} \cdot \frac{1}{\sqrt{r}} \cdot r |\phi\rangle = |\phi\rangle}}
 \end{aligned}$$

If we run QPE, the output will be  $s/r$  for one of these states.

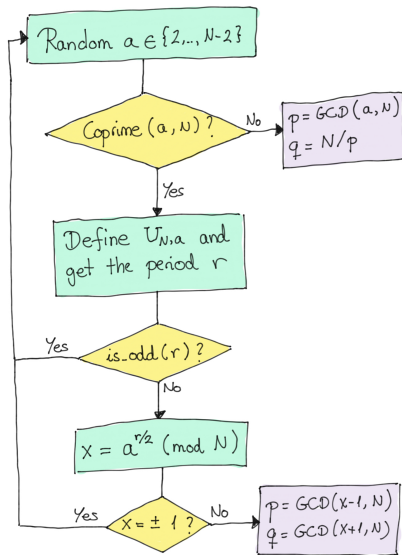
# Overview

Shor's algorithm is used to factor some number  $N$  into

where  $p$  and  $q$  are prime.

A quantum computer runs order finding, and the result is used to obtain  $p$  and  $q$ .

The rest of the algorithm is classical.



## Non-trivial square roots

Idea: find a *non-trivial square root* of  $N$ , i.e., some  $x \neq \pm 1$  s.t.

If we find such an  $x$ , then we know

This means that

for some integer  $k$ .



## Non-trivial square roots

If

then  $x - 1$  is a multiple of one of  $p$  or  $q$ , and  $x + 1$  is a multiple of the other. If

we can compute the values of  $p$  and  $q$  by finding their *gcd* with  $N$ :

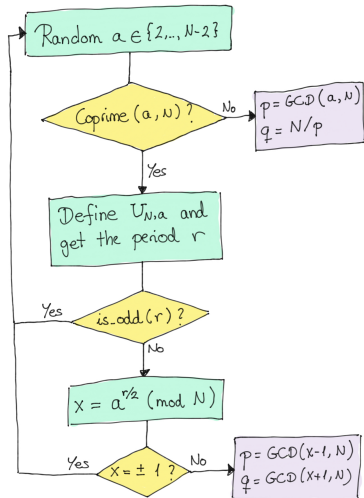
But... how do we find such an  $x$ ?

## Non-trivial square roots and factoring

It's actually okay to find any *even* power of  $x$  for which this holds:

We can use order finding to find such an  $r$ , and it is an even number, then we can find an  $x$  and factor  $N$ .

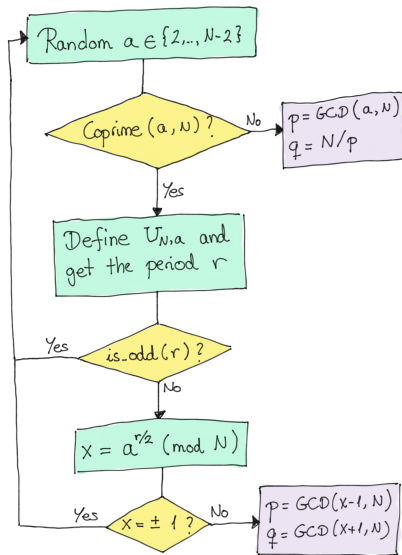
# Shor's algorithm



Is this really efficient?

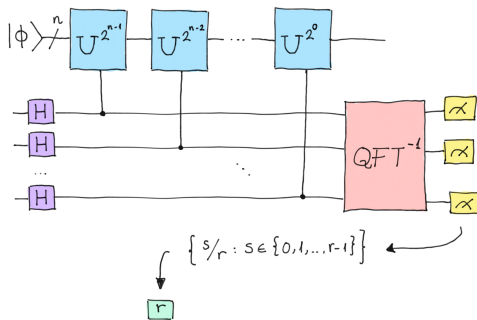
**GCD:** polynomial w/Euclid's algorithm

**Modular exponentiation:** can use exponentiation by squaring, other methods to reduce number of operations and memory required



# Is this really efficient?

Quantum part: let  $L = \lceil \log_2 N \rceil$ .



**QFT:** polynomial in number of qubits  $O(L^2)$

**Controlled- $U$  gates:** implemented using something called *modular exponentiation* in  $O(L^3)$  gates.

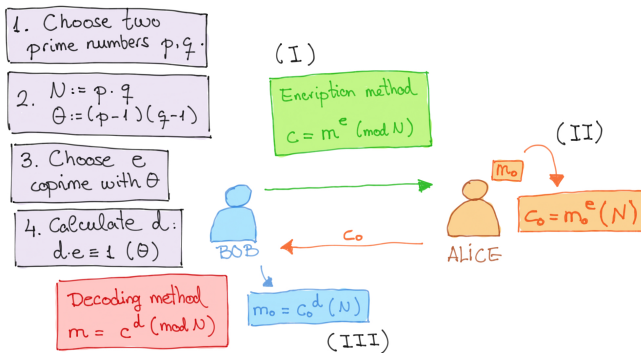
# Public-key cryptosystems

Two different types of cryptosystems:

- Symmetric: the key used to decode is the same as (or can easily be obtained from) the one used to encode
- Asymmetric / public-key: the key used to decode is different than the one used to encode

Both are used in modern infrastructure and each has its own advantages/disadvantages, attack vectors, and vulnerabilities.

RSA (Rivest–Shamir–Adleman) is a public-key cryptosystem.



## RSA: brief review of number theory concepts

The math behind RSA involves **modular arithmetic** and a few other number-theoretic ideas:

**Greatest common divisor (gcd):**  $\gcd(a, b)$  is the largest integer factor that divides perfectly into both  $a$  and  $b$



## RSA: brief review of number theory concepts

The math behind RSA involves **modular arithmetic** and a few other number-theoretic ideas:

**Greatest common divisor (gcd):**  $\gcd(a, b)$  is the largest integer factor that divides perfectly into both  $a$  and  $b$

**Co-prime:**  $a$  and  $b$  are co-prime if  $\gcd(a, b) = 1$ .

## RSA: brief review of number theory concepts

The math behind RSA involves **modular arithmetic** and a few other number-theoretic ideas:

**Greatest common divisor (gcd):**  $\gcd(a, b)$  is the largest integer factor that divides perfectly into both  $a$  and  $b$

**Co-prime:**  $a$  and  $b$  are co-prime if  $\gcd(a, b) = 1$ .

**Modular inverse:** Given a number  $a$  and modulus  $N$ , the inverse of  $a$  is the integer  $b$  such that  $ab \equiv 1 \pmod{N}$ . This exists *only if*  $a$  and  $N$  are co-prime.

## RSA: brief review of number theory concepts

The math behind RSA involves **modular arithmetic** and a few other number-theoretic ideas:

**Greatest common divisor (gcd):**  $\gcd(a, b)$  is the largest integer factor that divides perfectly into both  $a$  and  $b$

**Co-prime:**  $a$  and  $b$  are co-prime if  $\gcd(a, b) = 1$ .

**Modular inverse:** Given a number  $a$  and modulus  $N$ , the inverse of  $a$  is the integer  $b$  such that  $ab \equiv 1 \pmod{N}$ . This exists *only if*  $a$  and  $N$  are co-prime.

**Fermat's little theorem:** if  $p$  prime and  $a$  is an integer, then  $a^p \equiv a \pmod{p}$ . If  $a$  is not divisible by  $p$ , then  $a^{p-1} \equiv 1 \pmod{p}$ .

## Step 1

Choose two *prime numbers*,  $p$  and  $q$ .

## Step 2

Compute:

$$N = p \cdot q$$

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

## Step 3

Choose a value  $e$  that is *co-prime* with  $\theta$ .

## Step 4

Compute the inverse of  $e \bmod \theta$ , i.e., find  $d$  s.t.

$$d \cdot e \equiv 1 \bmod \theta$$

The *public key* is the pair  $(e, N)$ .

The *private key* is the pair  $(d, N)$ .

## Encoding

Acquire the public key  $(e, N)$  of the party you wish to send something to. To send the message  $m$ , encode it as

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

## Decoding

If you receive  $c$  and have the private key  $(d, N)$ , decode like so:

$$c^d \bmod N = (m^e)^d \bmod N = m$$

Two cases to consider to understand why this works.

Since  $ed = 1 \bmod \theta$ , there exists integer  $k$  such that  $ed = 1 + k\theta$ .

**Case 1:**  $m$  co-prime with  $N$

It is a known result in number theory that if  $m$  and  $N$  are co-prime, then  $m^\theta \equiv 1 \bmod N$  where  $\theta = (p-1)(q-1)$ . Thus,

**Case 2:**  $m$  not co-prime with  $N$

Then,  $\gcd(m, N) > 1$ . Must be  $p$  or  $q$ , because those are the only two factors of  $N$ .

Suppose  $\gcd(m, N) = p$ . Then,  $p$  also divides  $m$ ,

But  $q$  does not.  $q$  is prime, so  $\gcd(q, m) = 1$ . So by Fermat's little theorem,



**Case 2:**  $m$  not co-prime with  $N$

Again, since  $ed = 1 \bmod \theta$ , there exists  $k$  such that  $ed = 1 + k\theta$ .

So we have

It follows that

# RSA and factoring

- To decrypt the message, we must learn  $d$
- We know  $e$ , and that  $de \equiv 1 \pmod{\theta}$

We have only  $e$ , and  $N$ .

However,  $N$  and  $\theta$  are based on the same two prime numbers:

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

If we can factor  $N = pq$ , we find  $\theta$  and can decode the message!

The security of RSA relies on the fact that factoring for large numbers is computationally intractable.

# Computational complexity of breaking RSA

Current recommended key ( $N$ ) sizes are 2048- and 4096-bit.

The largest key size cracked so far is **829 bits** in 2020. See:  
[https://en.wikipedia.org/wiki/RSA\\_numbers](https://en.wikipedia.org/wiki/RSA_numbers)

Best-known classical algorithm:

## General number field sieve

---

From Wikipedia, the free encyclopedia

In **number theory**, the **general number field sieve (GNFS)** is the most **efficient** classical **algorithm** known for **factoring integers** larger than  $10^{100}$ . **Heuristically**, its **complexity** for factoring an integer  $n$  (consisting of  $\lfloor \log_2 n \rfloor + 1$  bits) is of the form

$$\exp\left(\left(\sqrt[3]{\frac{64}{9}} + o(1)\right) (\ln n)^{\frac{1}{3}} (\ln \ln n)^{\frac{2}{3}}\right) = L_n \left[\frac{1}{3}, \sqrt[3]{\frac{64}{9}}\right]$$

# Computational complexity of breaking RSA

On a quantum computer, there exists an algorithm that can help us solve the problem in *polynomial time*.

But it is still going to be a long time before that happens.

RSA-2048	Old estimates				Current estimates			
$p_g$	$n_\ell$	$n_p$	quantum resources	time	$n_\ell$	$n_p$	quantum resources	time
$10^{-3}$	6190	19.20	1.17	1.46	8194	22.27	0.27	0.34
$10^{-5}$	6190	9.66	0.34	0.84	8194	8.70	0.06	0.15

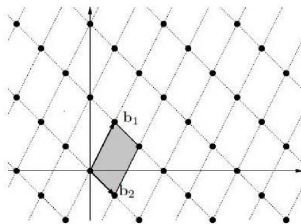
**Table 2.** RSA-2048 security estimates. Here  $n_\ell$  denotes the number of logical qubits,  $n_p$  denotes the number of physical qubits (in millions), time denotes the expected time (in hours) to break the scheme, and *quantum resources* (*quantum resources*) are expressed in units of megaqubitdays. The corresponding classical security parameter is 112 bits.

Image: V. Gheorghiu and M. Mosca, *A resource estimation framework for quantum attacks against cryptographic functions - recent developments*. Feb. 15 2021.

# What can / should we do about this?

RSA will not be cracked tomorrow. BUT, it's only a matter of time. We need to use this time wisely to re-tool our infrastructure. One option: **post-quantum cryptography**.

**Figure 2** A 2-dimensional lattice generated by the basis  $B = [b_1, b_2]$



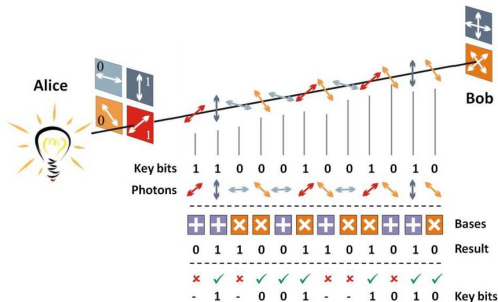
THE LWE problem was formally defined by Regev (2005). It

Use hard problems that *don't* have (known) efficient quantum solutions.

Image credit: G. Zhang, J. Qin. *Lattice-based threshold cryptography and its applications in distributed cloud computing*. Int. J. High Perform. Comput. Netw. 2015

# What can / should we do about this?

Symmetric crypto remains largely secure; use **quantum key distribution** to perform key exchange for it rather than RSA.



Theoretically secure, but can be challenging to implement, and other potential attack vectors such as hardware.

Image credit: Carrasco-Casado, Marmol, Denisenko. (2016) *Free-Space Quantum Key Distribution*. Optical Wireless Communications - An Emerging Technology (pp.589-607)

# What can / should we do about this?

Ethical dilemma: if we know quantum computers have malicious applications like breaking our cryptography infrastructure

- Should we still build them?
- Who should get to build them?
- Who should get to *use* them?

# What can / should we do about this?

Ethical dilemma: if we know quantum computers have malicious applications like breaking our cryptography infrastructure

- Should we still build them?
- Who should get to build them?
- Who should get to *use* them?



# Next time

## Content:

- Moving back to variational algorithms

## Action items:

1. Work on A2 once available
2. Start working on prototype implementation for project

## Recommended reading:

- Codebook nodes S.1-S.5
- Nielsen & Chuang 5.3, Appendix A.5