Quantum computers and cryptography

PCMI 2022 Undergraduate Summer School

Lecture 5

Christelle Vincent

University of Vermont

July 25, 2022

Our slogan

The fact that makes public key cryptography possible is that there are mathematical operations that are **easy** to do and **hard** to undo.

For example

RSA encryption:

- private key: the two primes 1489 and 701
- public key: their product 1,043,789.

We can share the public key, since factoring is **hard** (if we don't have a quantum computer).

Quick computational complexity review

A problem is **hard** if it can only be solved in exponential time.

It is **easy** if we have an algorithm to solve it in polynomial time.

Hardness of problems over time

Newer computers do the same thing but faster.

Cryptographic parameters are updated to keep up with technology.

	then	now
RSA-100	few days (1991)	72 mins (2012)
RSA-110	one month (1992)	4 hours (2012)

Enter quantum computers

Quantum computers do **something completely different**.

When we can use their properties, a hard problem can become easy.

The punchline

Shor's algorithm running on a quantum computer:

- factoring: from subexponential to polynomial
- DLP: from exponential to polynomial

The majority of the security of the internet depends on the hardness of these problems :(

What we mean: RSA e.g.

Currently it is recommended to use RSA with a 3072-bit modulus.

Post-quantum, RSA should be secure with a 1TB modulus which is the product of 2^{31} 4096-bit primes.

(Encryption takes 10 hours, decryption ??)

Today

- How are quantum computers so fast??
- Can we still have cryptography??

How are quantum computers so fast??

Shor: how to compute the **period** of a function in quantum polynomial time.

It turns out that this is enough to factor and solve the DLP.

Solving DLP: Where is the period?

Let $G = \langle g \rangle$ be a cyclic group of order n.

Let $h = g^x$ for some (unknown) x.

Consider the function

$$\begin{aligned} f\colon C_n \times C_n &\to G \\ (a, b) &\mapsto g^a h^{-b} = g^{a-bx}. \end{aligned}$$

Solving DLP: Where is the period?

$$\begin{aligned} f\colon C_n \times C_n &\to G \\ (a,b) &\mapsto g^a h^{-b} = g^{a-bx}. \end{aligned}$$

We have then that

$$f(a_1, b_1) = f(a_2, b_2)$$

if and only if

$$(a_2, b_2) = (a_1, b_1) + \lambda(x, 1) \quad \text{for some } \lambda.$$

Therefore finding x reduces to finding the period of f.

Factoring: Where is the period?

This is more complicated.

First assume that N is odd and not a power of a prime.

We begin with a random number $a < N$ with $\gcd(a, N) = 1$.
(Otherwise we are done.)

If at any point our assumption is false, we start over with a new a.
(There is a 50% chance of success.)

Factoring: Where is the period?

1. Compute the multiplicative order r of $a \pmod{N}$.
   (This is the period of $f(x) = a^x \pmod{N}$!)

2. Assuming r is even, compute $a^{r/2} \pmod{N}$.

3. Assuming $a^{r/2} \not\equiv -1 \pmod{N}$, then

$$\gcd(a^{r/2}+1, N) \quad \text{and} \quad \gcd(a^{r/2}-1, N)$$

   are nontrivial factors of N and we are done.

Wait, what?

Notice that

$$(a^{r/2}+1)(a^{r/2}-1) = a^r - 1 \equiv 0 \pmod{N}.$$

Therefore there is an integer k with

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

But N doesn't divide $(a^{r/2}+1)$ nor $(a^{r/2}-1)$.

Factoring: There is the period!

Factoring is reduced to computing the period of

$$f(x) = a^x \pmod{N}.$$

Quantum computers lightning fast

In a classical computer, an n-bit register contains an n-bit number.

For example, a 2-bit register can contain **either**

$$00 \quad \text{or} \quad 01 \quad \text{or} \quad 10 \quad \text{or} \quad 11.$$

What is a qubit?

An n-qubit register contains **a superposition** of n-bit numbers.

For example, a 2-qubit register contains a superposition

$$x_0|00\rangle + x_1|01\rangle + x_2|10\rangle + x_3|11\rangle,$$

where the x_is are complex numbers and

$$|x_0|^2 + |x_1|^2 + |x_2|^2 + |x_3|^2 = 1.$$

How do qubits speak to us?

To obtain an answer, we measure the superposition

$$x_0|00\rangle + x_1|01\rangle + x_2|10\rangle + x_3|11\rangle,$$

and observe the output $|i\rangle$ with probability $|x_i|^2$.

Idea: Manipulate the qubit so the answer $|i\rangle$ is observed with high probability.

How do we manipulate qubits?

Operations on qubits must be **reversible**.

In fact, all operations on qubits are given by **unitary matrices**.

(These are invertible matrices that preserve the property that

$$\sum_{i=0}^{2^n} |x_i|^2 = 1.)$$

Fun consequence/An example

Suppose that I have two 1-qubit registers

$$|a\rangle \quad \text{and} \quad |b\rangle$$

and I want to compute their sum.

Then I must compute

$$|a, a+b\rangle$$

so that the addition operation is reversible!

Ok, but how?

Addition is done with the CNOT gate, given by the matrix

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

If I want to compute $a + b$, then I can make the superposition

$$|a, b\rangle$$

and apply this gate to it.

Ok, but how?

Indeed, the superposition

$$x_0|00\rangle + x_1|01\rangle + x_2|10\rangle + x_3|11\rangle,$$

is sent to

$$x_0|00\rangle + x_1|01\rangle + x_3|10\rangle + x_2|11\rangle,$$

by this matrix.

When I read the answer,

- the first qubit is a
- and the second qubit is $a + b$.

Quantum Fourier transform

Consider an n-qubit register containing the superposition

$$x_0|0\dots0\rangle + x_1|0\dots1\rangle + x_{2^n-1}|1\dots1\rangle = \sum_{i=0}^{2^n-1} x_i|i\rangle.$$

Quantum Fourier transform

Then the Fourier transform is given by the matrix

$$F_{2^n} = \frac{1}{\sqrt{2^n}} \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \zeta_{2^n} & \zeta_{2^n}^2 & \dots & \zeta_{2^n}^{2^n-1} \\ 1 & \zeta_{2^n}^2 & \zeta_{2^n}^4 & \dots & \zeta_{2^n}^{2(2^n-1)} \\ & & & \dots & \\ 1 & \zeta_{2^n}^{2^n-1} & \zeta_{2^n}^{2(2^n-1)} & \dots & \zeta_{2^n}^{(2^n-1)(2^n-1)} \end{pmatrix},$$

where ζ_{2^n} is a primitive 2^nth root of unity.

Quantum Fourier transform

Specifically, the new superposition is given by

$$\sum_{i=0}^{2^n-1} y_i |i\rangle$$

where

$$F_{2^n} x = y.$$

Quantum Fourier transform

Another way to write this is that the new superposition is given by

$$\sum_{i=0}^{2^n-1} y_i |i\rangle$$

where

$$y_i = \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} x_k \zeta_{2^n}^{ik}.$$

Back to the periods

For simplicity we show how to find the period of the function

$$f(x) = a^x \pmod N.$$

We begin by picking q such that

$$N^2 \leq 2^q < 2N^2.$$

This guarantees that there are at least N different values between 0 and $2^q - 1$ such that

$$a^{x_1} \equiv a^{x_2} \pmod N.$$

Shor's algorithm: High level overview

To find the multiplicative order r of $a \pmod{N}$:

1. Superpose the values $a^x \pmod{N}$ for $0 \leq x \leq 2^q - 1$.
2. Manipulate the superposition to measure an integer y such that $\frac{y}{2^q}$ is very close to a fraction with denominator r.
3. Use continued fraction expansions to find the denominator of that fraction.

The algorithm

1. Begin with the superposition

$$\frac{1}{\sqrt{2^q}} \sum_{i=0}^{2^q-1} |i\rangle.$$

2. Construct a^x as a quantum function and apply it to the superposition to get

$$\frac{1}{\sqrt{2^q}} \sum_{i=0}^{2^q-1} |i, a^i\rangle.$$

The algorithm

Recall that our register now contains

$$\frac{1}{\sqrt{2^q}} \sum_{i=0}^{2^q-1} |i, a^i\rangle$$

3. Apply the Fourier transform to the **inputs** only. This leads to the final state

$$\frac{1}{2^q} \sum_{i=0}^{2^q-1} \sum_{j=0}^{2^q-1} \zeta_{2^q}^{ij} |j, a^i\rangle = \frac{1}{2^q} \sum_{k=0}^{2^q-1} \sum_{j=0}^{2^q-1} |j, k\rangle \sum_{i: a^i = k} \zeta_{2^q}^{ij}.$$

The algorithm

4. Now measure the superposition. The probability of observing $|j, k\rangle$ is

$$\left| \frac{1}{2^q} \sum_{i : a^i = k} \zeta_{2^q}^{ij} \right|^2 = \frac{1}{2^{2q}} \left| \sum_{b : i_0 + rb < 2^q} \zeta_{2^q}^{(i_0 + rb)j} \right|^2$$
$$= \frac{1}{2^{2q}} \left| \sum_{b : i_0 + rb < 2^q} \left(\zeta_{2^q}^{rj} \right)^b \right|^2,$$

where i_0 is the smallest i with $a^i = k$.
This sum is greatest when $\zeta_{2^q}^{rj}$ is closest to 1.

The algorithm

5. Therefore with high probability, $\frac{rj}{2^q} \approx c$ with $c \in \mathbb{Z}$. Then

$$\frac{j}{2^q} \approx \frac{c}{r}.$$

To find $\frac{c}{r}$, look for some fraction $\frac{d}{s}$ with

1. $s < N$ and
2. $\left| \frac{j}{2^q} - \frac{d}{s} \right| < \frac{1}{2^{q+1}}$

With high probability, s is r or a factor of r.

The algorithm

6. Check if $a^s \equiv 1 \pmod{N}$,
   or try multiples of s,
   or start over, possibly with another value of a.

An example

Suppose we want to compute the order of 2 modulo 33. (It's 10.) Here $2^q = 2048$, so we compute the convergents of $\frac{j}{2048}$.

The algorithm will output the following:

j	Probability	s	j	Probability	s
0	10%	1	1024	10%	2
205	8.8%	10	1229	8.8%	5
409	2.5%	5	1433	2.5%	10
410	5.7%	5	1434	5.7%	10
614	5.7%	10	1638	5.7%	5
615	2.5%	10	1639	2.5%	5
819	8.8%	5	1843	8.8%	10

The probability of success is 68%.

Can we still have cryptography??

The upshot:
Don't know how to solve **every** problem with a quantum computer.

So: all we need are different problems!

Post-quantum vs quantum

- **Post-quantum cryptography** refers to ciphers that will be secure in a post-quantum world: Based on problems that are hard for a classical **and** a quantum computer.
- Only the attacker needs a quantum computer.
  The encryption/decryption takes place on a classical computer.

Post-quantum vs quantum

This is in contrast to **quantum cryptography**, which uses quantum phenomena to secure the information.

Post-quantum algorithm families

1. hash-based
2. code-based
3. multivariate
4. lattice-based
5. isogeny-based

Code-based ciphers

A **linear code** in mathematics is a subspace C of $\mathbb{F}_q^n$.

For example: $C = \{(0,0,0),(1,1,1)\} \subset \mathbb{F}_2^3$.

The "extra room" in $\mathbb{F}_q^n$ allows to correct errors.

Code-based ciphers

First proposed by McEliece in 1978:

- encryption is introducing errors, and
- decryption is correcting the errors.

Multivariate ciphers

Rely on difficulty of solving systems of quadratic multivariate equations over finite fields.

First proposed by Matsumoto and Imai in 1988.

Most basic cipher

1. Pick an easily invertible quadratic map $\mathcal{F}\colon \mathbb{F}_q^n \to \mathbb{F}_q^m$
2. Pick two linear transformations $S\colon \mathbb{F}_q^m \to \mathbb{F}_q^m$ and $T\colon \mathbb{F}_q^n \to \mathbb{F}_q^n$
3. Publish the composition $\mathcal{P} = S \circ \mathcal{F} \circ T$

Lattice-based ciphers

Rely on the difficulty of finding a short vector given a bad basis.

First proposed by Ajtai and Hoffstein-Pipher-Silverman in 1996.

This is what we will study for the rest of our time together.

Isogeny-based ciphers

Rely on the difficulty of navigating the isogeny graph of supersingular elliptic curves.

Key mathematical property: These graphs are expander graphs.

Charles-Goren-Lauter proposed a hash function in 2006 and De Feo, Jao and Plut proposed SIDH in 2011.

Thank you!