Last Time:

- Superdense Coding
- Entangled states on more than 2 qubits
- Started talking about Shor's factoring algorithm
  - Motivation: want efficient factoring algorithm to break RSA encryption

Today:

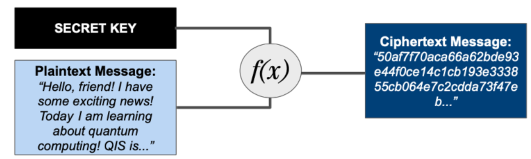
- Study factoring problem in more detail
- Understand the **non-quantum** part of Shor's algorithm

1

Securing Information with RSA Encryption

Makes data appear completely random unless viewed by intended recipient

If encryption key factors are unknown, data cannot be decrypted without significant time or computer resources



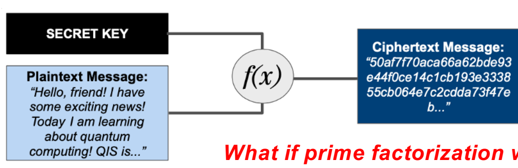
2

2

Securing Information with RSA Encryption

Relies on the difficulty of factoring the product of two large prime numbers

Multiplying is easy, but factoring ***seems*** very hard!



What if prime factorization was easier?

3

3

Quantum Factoring with Shor's Algorithm

Developed in 1994 by Peter Shor

- "Algorithms for quantum computation: discrete logarithms and factoring"
- "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer"

Exponential speedup compared to classical techniques!

First demonstration of significant quantum speedup for a practical application

4

4

Factoring Problem

Notation: integers x, r, d

$x \equiv r \bmod d$ means $x = r + dm$, for some integer m

$d|x$ means d divides x (i.e. $x \equiv 0 \bmod d$)

a non-trivial divisor of x is an integer d, where $d|x$ and $1 < d < x$

(1 and x are trivial divisors)

Factoring Problem

Input: composite integer x

Output: a non-trivial divisor of x

THE UNIVERSITY OF CHICAGO EPiQC UC SANTA BARBARA 5

5

Simple but Inefficient Factoring Algorithm: Trial Division

For $d = 2$ to floor$(\sqrt{x})$

If $d|x$

Return d

Correctly outputs non-trivial divisor, but what is running time on input x that is $n = ceil(\log x)$ bits long

For each d, can check if $d|x$ in polynomial time ($n^{O(1)}$ steps)

But, in worst case, need to try $\approx 2^{\frac{n}{2}}$ values of d

Overall, algorithm requires **exponential time**

Ex: To factor a 1000-bit number, need to try $\approx 2^{500}$ values of d (> # of atoms in the universe)

Breaking (modern) RSA involves factoring numbers that are thousands of bits long

Exponential time is **not practical!**

THE UNIVERSITY OF CHICAGO EPiQC UC SANTA BARBARA 6

6

Can we Factor More Efficiently?

Worst-cast running time

Trial division: $2^{O(n)}$ exponential time

Quadratic Sieve: better (but still exponential) bound $2^{O(\sqrt{n})}$

Uses elementary number theory

General Number Field Sieve: Slightly subexponential time (but still superpolynomial)

Uses not-so-elementary number theory (and an unproven but reasonable conjecture)

Still wildly impractical

No **known** classical algorithm can factor in polynomial time

Shor's quantum factoring algorithm runs in polynomial time

Uses elementary number theory similar to the Quadratic Sieve

Only quantum part: determining the period of a function using QFT (quantum Fourier transform)

THE UNIVERSITY OF CHICAGO EPiQC UC SANTA BARBARA 7

7

Real Experimental Results

Current record for largest (general) number that has been factored

Classical supercomputer: 829 bits

Quantum computer: the number 21

Not a 21-bit number… just $21 = 3 \cdot 7$

Fundamental issue: real quantum computers accumulate errors very quickly

THE UNIVERSITY OF CHICAGO EPiQC UC SANTA BARBARA 8

8

Shor's Algorithm = (a little) Number Theory + QFT

Say integer a is *useful* for x if $a^2 \equiv 1 \bmod x$, $a \not\equiv 1 \bmod x$, and $a \not\equiv -1 \bmod x$

Claim 1: If a is *useful* for x, then $\gcd(a-1, x)$ and $\gcd(a+1, x)$ are both non-trivial divisors of x
  $\gcd(w, x)$ is greatest common divisor of w and x (largest d where $d|w$ and $d|x$)

Claim 2: Can compute $\gcd(w, x)$ in polynomial time (even on classical computer)
  Using Euclid's algorithm

Claim 3: Can find such an a by computing period of certain function
  (ignoring certain trivial special cases for x)

Claim 4: Can compute period of desired function in polynomial time on a **quantum** computer

Above 4 claims ⇒ quantum polynomial time factoring algorithm

1: Useful Values Produce Non-Trivial Divisors

Suppose a is *useful* for x: $a^2 \equiv 1 \bmod x$, $a \not\equiv 1 \bmod x$, and $a \not\equiv -1 \bmod x$

Claim: $\gcd(a-1, x)$ is a non-trivial divisor of x
  By definition, $\gcd(a-1, x)\,|x$
  $\gcd(a-1, x) \neq x$
    If $\gcd(a-1, x) = x$, then $x|a-1 \Rightarrow a \equiv 1 \bmod x$; however, $a \not\equiv 1 \bmod x$
  $\gcd(a-1, x) \neq 1$
    $(a-1)(a+1) \equiv (a^2-1) \equiv 0 \bmod x \Rightarrow x|(a-1)(a+1)$
    If $\gcd(a-1, x) = 1$, then $x|a+1 \Rightarrow a \equiv -1 \bmod x$; however, $a \not\equiv -1 \bmod x$

By an analogous argument, so is $\gcd(a+1, x)$

2: Euclid's Algorithm Computes GCD in Polynomial Time

```
GCD(w,x)
    large=max(w,x)
    small=min(w,x)
    r=large mod small
    if r==0
        return small
    return GCD(r,small)
```

Very efficient: Runs in time $O(n)$, $n = \max(\log(w), \log(x))$

3: Period-finding Produces Useful Values

Consider integer $b \in [2, x-1]$ where $\gcd(x, b) = 1$
  Let T denote period of function $f(y) = b^y \bmod x$: is smallest positive integer s.t.
    $f(y) = f(y+T)$ for all y
    Equivalently, $b^T \equiv 1 \; mod \; x$ $(f(y+T) \equiv b^{y+T} \equiv b^y b^T \bmod x)$
  Claim: If T is even and $b^{\frac{T}{2}} \not\equiv -1 \bmod x$, then $a = b^{\frac{T}{2}}$ is useful for x
    $a^2 \equiv b^T \equiv 1 \bmod x$
    $a \not\equiv -1 \bmod x$ (by def)
    $a \not\equiv 1 \bmod x$ (o.w. $b^{\frac{T}{2}} \equiv 1 \; mod \; x$, contradicts fact that T is period)

For simplicity, assume $x = pq$ for distinct odd primes p and q (can extend to general case, conveys main idea)
  Claim: if randomly generate $b \in [2, x-1]$ s.t. $\gcd(x, b) = 1$
    then with probability $\geq \frac{3}{8}$, T is even and $b^{\frac{T}{2}} \not\equiv -1 \bmod x$
    Proof uses more number theory, will skip

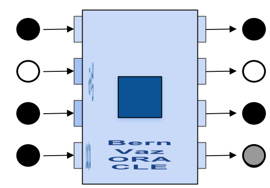
3: Period-finding Produces Useful Values

Algorithm Factor(x)
- Randomly generate $b \in [2, x-1]$
- If $\gcd(x, b) \neq 1$
  - Return $\gcd(x, b)$ (is non-trivial divisor of x)
- Else
  - T = period of function $f(y) = b^y \bmod x$
    - If T is even and $b^{\frac{T}{2}} \not\equiv -1 \bmod x$ (happens with prob $\geq \frac{3}{8}$)
      - $a = b^{\frac{T}{2}}$ (is useful for x)
      - Return $\gcd(a-1, x)$ (is non−trivial divisor of x)
    - Else Return Factor(x)

After $O(1)$ expected runs of algorithm, get non-trivial factor
- Each run takes time poly(n) + time needed to find T
- If can find T in polynomial time, can factor in polynomial time

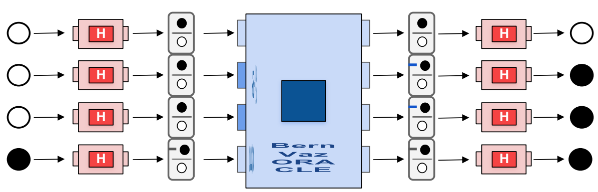
Recall: BernVaz Oracle

There exists a secret 3-bit code.
The oracle contains a sequence of C-NOT gates in which each input corresponding to a 1 in the secret code is the control for the response, which is the target.



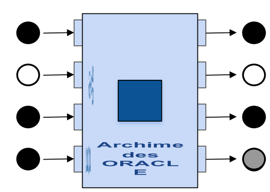
Recall: BernVaz Simultaneous algorithm

What is the secret code?
- Input all white balls for guess, black ball for response.
- Put H-gates before and after query to oracle.
- Exploit phase flip to make response bit flip C-NOT-connected input bits.
- **Output will be identical to secret code**



Recall Archimedes Oracle

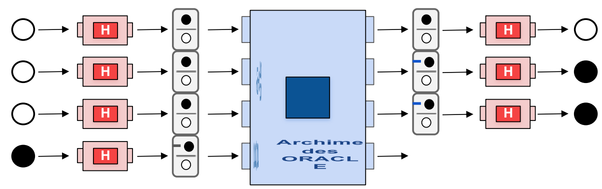
There exists a set of 3-bit codes.
Given a 3-bit guess, the oracle will **flip the response** if the guess is one of the 3-bit codes.



Recall" Archimedes Simultaneous algorithm

There are either 0 or 4 secret codes. Which is it?

Input all white balls for guess, black ball for response.
Put H-gates before and after query to oracle.
Exploit interference of responses
Output will be all whites if 0, some non-white if 4



This material is based upon work supported by the National Science Foundation under Grants No. 1730088 and No. 1730449. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.