Cracking RSA with Quantum Computing

Max Ovsiankin

May 9, 2018

Outline

1. The Setting
2. Classical Computers
3. Quantum Computers
4. Shor's Algorithm

The Setting

RSA is a commonly used set of algorithms that provides security when sending encrypted messages.

The Setting

RSA is a commonly used set of algorithms that provides security when sending encrypted messages.
Crucially, the security of RSA depends on the hardness of factoring a number $N = pq$, where p and q are large prime numbers ('secrecy').

The Setting

RSA is a commonly used set of algorithms that provides security when sending encrypted messages.
Crucially, the security of RSA depends on the hardness of factoring a number $N = pq$, where p and q are large prime numbers ('secrecy').

The Setting

An assumption of RSA security (that follows from $\mathrm{P} \neq \mathrm{NP}$) is that prime numbers cannot be factored with a polynomial-time algorithm in the number of bits of N.

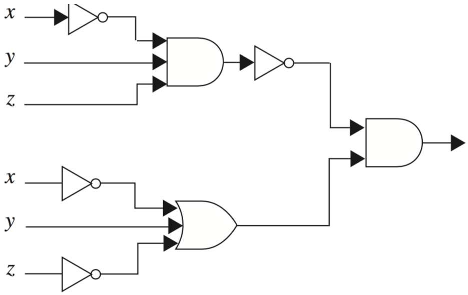
The Setting

An assumption of RSA security (that follows from $\mathrm{P} \neq \mathrm{NP}$) is that prime numbers cannot be factored with a polynomial-time algorithm in the number of bits of N.

Quantum computers *are* able to factor in polynomial time. This talk will focus on explaining how quantum algorithms work, building up to Shor's famous algorithm for factoring.

What Computers Look Like

Equivalent to a circuit whose representation can be quickly computed:



What Computers Look Like

We can think of a circuit as having n registers, each of which contain 0 or 1. A possible *state* of these n registers is an element of $\{0,1\}^n$ (n-length bitstring). Then the action of a gate can be described as a matrix of 0s and 1s.

What Computers Look Like: NOT gate

NOT gate:

$$\mathrm{NOT}(0) = 1$$
$$\mathrm{NOT}(1) = 0$$

What Computers Look Like: NOT gate

NOT gate:

$$\mathrm{NOT}(0) = 1$$
$$\mathrm{NOT}(1) = 0$$

Let's relabel $0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$, $1 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$.
Elements of a 2-element vector space, as there are 2 possibilities for bits.

What Computers Look Like: NOT gate

NOT gate:

$$\mathrm{NOT}\left(\begin{bmatrix}1\\0\end{bmatrix}\right) = \begin{bmatrix}0\\1\end{bmatrix}$$

$$\mathrm{NOT}\left(\begin{bmatrix}0\\1\end{bmatrix}\right) = \begin{bmatrix}1\\0\end{bmatrix}$$

This strongly suggests we can consider it as a matrix:

$$\mathrm{NOT} = \begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}$$

What Computers Look Like: AND gate

AND gate:

$$\begin{aligned}\mathrm{AND}\,(0,0) &= 0\\ \mathrm{AND}\,(0,1) &= 0\\ \mathrm{AND}\,(1,0) &= 0\\ \mathrm{AND}\,(1,1) &= 1\end{aligned}$$

What Computers Look Like: AND gate

For two input bits, there are $4 = 2^2$ possible states.

$$(0,0) = |00\rangle = |0\rangle \otimes |0\rangle$$

$$(0,1) = |01\rangle = |0\rangle \otimes |1\rangle$$

$$(1,0) = |10\rangle = |1\rangle \otimes |0\rangle$$

$$(0,1) = |11\rangle = |1\rangle \otimes |1\rangle$$

What Computers Look Like: AND gate

$$\mathrm{AND}\left(|00\rangle\right) = |0\rangle$$

$$\mathrm{AND}\left(|01\rangle\right) = |0\rangle$$

$$\mathrm{AND}\left(|10\rangle\right) = |0\rangle$$

$$\mathrm{AND}\left(|11\rangle\right) = |1\rangle$$

What Computers Look Like: AND gate

$$\mathrm{AND}\left(1\left|00\right\rangle + 0\left|01\right\rangle + 0\left|10\right\rangle + 0\left|11\right\rangle\right) = 1\left|0\right\rangle + 0\left|1\right\rangle$$

$$\mathrm{AND}\left(0\left|00\right\rangle + 1\left|01\right\rangle + 0\left|10\right\rangle + 0\left|11\right\rangle\right) = 1\left|0\right\rangle + 0\left|1\right\rangle$$

$$\mathrm{AND}\left(0\left|00\right\rangle + 0\left|01\right\rangle + 1\left|10\right\rangle + 0\left|11\right\rangle\right) = 1\left|0\right\rangle + 0\left|1\right\rangle$$

$$\mathrm{AND}\left(0\left|00\right\rangle + 0\left|01\right\rangle + 0\left|10\right\rangle + 1\left|11\right\rangle\right) = 0\left|0\right\rangle + 1\left|1\right\rangle$$

What Computers Look Like: AND gate

$$\begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} (1\left|00\right\rangle + 0\left|01\right\rangle + 0\left|10\right\rangle + 0\left|11\right\rangle) = 1\left|0\right\rangle + 0\left|1\right\rangle$$

$$\begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} (0\left|00\right\rangle + 1\left|01\right\rangle + 0\left|10\right\rangle + 0\left|11\right\rangle) = 1\left|0\right\rangle + 0\left|1\right\rangle$$

$$\begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} (0\left|00\right\rangle + 0\left|01\right\rangle + 1\left|10\right\rangle + 0\left|11\right\rangle) = 1\left|0\right\rangle + 0\left|1\right\rangle$$

$$\begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} (0\left|00\right\rangle + 0\left|01\right\rangle + 0\left|10\right\rangle + 1\left|11\right\rangle) = 0\left|0\right\rangle + 1\left|1\right\rangle$$

What Computers Look Like: AND gate

Okay, we have a vector space now. What about linear combinations?

$$\begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \left(\frac{1}{2} |00\rangle + 0 |01\rangle + \frac{1}{4} |10\rangle + \frac{1}{4} |11\rangle \right) = ???$$

What changes for Quantum?

The state of an n-bit classical computer is a vector in $\mathbb{R}^{2^n}$ with only one coefficient nonzero. (we already saw this in explaining classical computers)

What changes for Quantum?

The state of an n-bit classical computer is a vector in $\mathbb{R}^{2^n}$ with only one coefficient nonzero. (we already saw this in explaining classical computers)

The state of an n-*qu*bit quantum computer is a vector in $\mathbb{C}^{2^n}$ that is normalized (this reflects the underlying quantum property of superposition):

$$a_0 \left|00\right\rangle + a_1 \left|01\right\rangle + a_2 \left|10\right\rangle + a_3 \left|11\right\rangle$$

with $a_i \in \mathbb{C}$ and $\sum_{i=0}^{n-1} |a_i|^2 = 1$ (unit vectors)

What changes for Quantum?

The operation we perform on a n-qubit 'register' is to measure it.

What changes for Quantum?

The operation we perform on a n-qubit 'register' is to measure it.

$$a_0 |00\rangle + a_1 |01\rangle + a_2 |10\rangle + a_3 |11\rangle$$

What changes for Quantum?

The operation we perform on a n-qubit 'register' is to measure it.

$$a_0 \left|00\right\rangle + a_1 \left|01\right\rangle + a_2 \left|10\right\rangle + a_3 \left|11\right\rangle$$

This produces $\left|00\right\rangle$ with probability $|a_0|^2$, $\left|01\right\rangle$ with probability $|a_1|^2$, etc.

Quantum Gates

Our quantum gates now take unit complex vectors to unit complex vectors (they are exactly the *unitary* matrices)!

Quantum Gates

Our quantum gates now take unit complex vectors to unit complex vectors (they are exactly the *unitary* matrices)!

Hadmard

$$\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}$$

Quantum Gates

Our quantum gates now take unit complex vectors to unit complex vectors (they are exactly the *unitary* matrices)!

Hadmard

$$\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\1 & -1\end{bmatrix}$$

c-NOT

$$\begin{bmatrix}1 & 0 & 0 & 0\\0 & 1 & 0 & 0\\0 & 0 & 0 & 1\\0 & 0 & 1 & 0\end{bmatrix}$$

Quantum Gates

Our quantum gates now take unit complex vectors to unit complex vectors (they are exactly the *unitary* matrices)!

c-NOT

Hadmard

Phase Rotation

$$\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}$$

$$\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\end{bmatrix}$$

$$\begin{bmatrix}1 & 0\\ 0 & e^{i\frac{\pi}{4}}\end{bmatrix}$$

Comparison to Classical

$$\mathrm{NOT}\left(1\left|0\right\rangle + 0\left|1\right\rangle\right) = 0\left|0\right\rangle + 1\left|1\right\rangle$$
$$\mathrm{NOT}\left(0\left|0\right\rangle + 1\left|1\right\rangle\right) = 1\left|0\right\rangle + 0\left|1\right\rangle$$

Comparison to Classical

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} (1\left|0\right\rangle + 0\left|1\right\rangle) = 1\left|0\right\rangle + 0\left|1\right\rangle$$

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} (0\left|0\right\rangle + 1\left|1\right\rangle) = 0\left|0\right\rangle + 1\left|1\right\rangle$$

Comparison to Classical

Superposition!

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \left(\frac{1}{\sqrt{2}} |0\rangle + \frac{1}{\sqrt{2}} |1\rangle \right) = \frac{1}{\sqrt{2}} (|0\rangle + |1\rangle)$$

Like applying the gate twice 'at the same time'!

c-NOT

Want a gate to 'conditionally' apply its effect. Control bit controls whether gate act, and the gate acts on the second bit.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} (|0\rangle \otimes |0\rangle) = |0\rangle \otimes |0\rangle$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} (|0\rangle \otimes |1\rangle) = |0\rangle \otimes |1\rangle$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} (|1\rangle \otimes |0\rangle) = |1\rangle \otimes |1\rangle$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} (|1\rangle \otimes |1\rangle) = |1\rangle \otimes |0\rangle$$

Phase estimation

PHASE-ESTIMATE

Input: $\omega \in [0, 1]$ unknown. We get a state of the form

$$|\psi\rangle = \frac{1}{\sqrt{2}^n} \sum_{i=0}^{2^n-1} e^{2\pi i \omega y} |y\rangle$$

Output: An estimate of ω.

Phase estimation

PHASE-ESTIMATE
Input: $\omega \in [0, 1]$ unknown. We get a state of the form

$$|\psi\rangle = \frac{1}{\sqrt{2}^n} \sum_{i=0}^{2^n-1} e^{2\pi i \omega y} |y\rangle$$

Output: An estimate of ω.
What does the output even look like? Need to approximate ω with n bits when measured.

Phase estimation

PHASE-ESTIMATE

Input: $\omega \in [0, 1]$ unknown. We get a state of the form

$$|\psi\rangle = \frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} e^{2\pi i \omega y} |y\rangle$$

Output: An estimate of $\omega = 0.x_1 x_2 \ldots x_n = \frac{x}{2^n}$ for $x \in \{0, 1, 2, \ldots, 2^n - 1\}$

Phase estimation

It turns out

$$\frac{1}{\sqrt{2}^n}\sum_{i=0}^{2^n-1} e^{2\pi i \omega y}|y\rangle = \frac{1}{\sqrt{2}^n}\left(|0\rangle + e^{2\pi i 0.x_n}|1\rangle\right)\otimes\left(|0\rangle + e^{2\pi i 0.x_{n-1}x_n}|1\rangle\right)$$

$$\otimes \dots$$

$$\otimes\left(|0\rangle + e^{2\pi i 0.x_2x_3\dots x_{n-1}}|1\rangle\right)\otimes\left(|0\rangle + e^{2\pi i 0.x_1x_2\dots x_n}|1\rangle\right)$$

Phase estimation

It turns out

$$\frac{1}{\sqrt{2}^n}\sum_{i=0}^{2^n-1} e^{2\pi i\omega y}|y\rangle = \frac{1}{\sqrt{2}^n}\big(|0\rangle + e^{2\pi i 0.x_n}|1\rangle\big)\otimes\big(|0\rangle + e^{2\pi i 0.x_{n-1}x_n}|1\rangle\big)$$

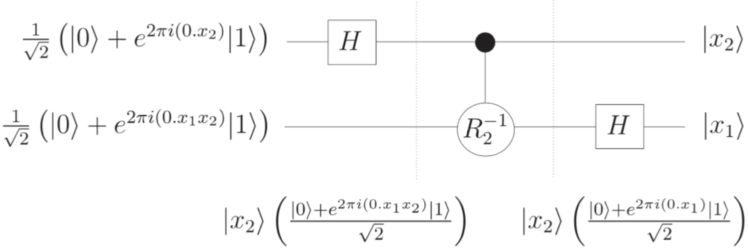
$$\otimes \dots$$

$$\otimes \big(|0\rangle + e^{2\pi i 0.x_2x_3\dots x_{n-1}}|1\rangle\big)\otimes\big(|0\rangle + e^{2\pi i 0.x_1x_2\dots x_n}|1\rangle\big)$$

Idea: 'pull' x_i out one by one, then remove them from the remaining bits.

Phase estimation

Implementation (we will examine what happens):



Phase estimation

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

Looking at

$$H \frac{1}{\sqrt{2}} (|0\rangle + e^{2\pi i (0.x_2)} |1\rangle) = H \frac{1}{\sqrt{2}} (|0\rangle + (-1)^{x_2} |1\rangle) = |x_2\rangle$$

Phase estimation

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

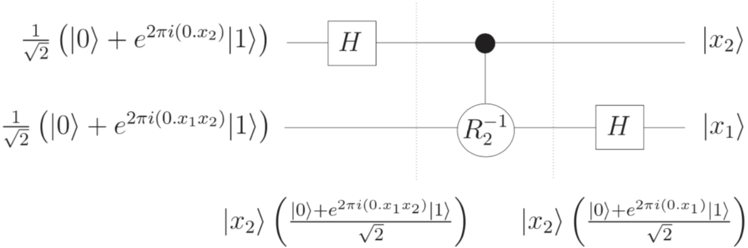
Looking at

$$H \frac{1}{\sqrt{2}} (|0\rangle + e^{2\pi i (0.x_2)} |1\rangle) = H \frac{1}{\sqrt{2}} (|0\rangle + (-1)^{x_2} |1\rangle) = |x_2\rangle$$

So H 'pulls' x_2 out for us, into a qbit!

Phase estimation

We have $|x_2\rangle$ after the first H, and we want to 'eliminate' it from the 2nd qubit:



Phase estimation

$$R_2^{-1} = \begin{bmatrix} 1 & 0 \\ 0 & e^{-2\pi i(0.01)} \end{bmatrix}$$

R_2 *only* rotates the phase on $|1\rangle$.
Action when $x_2 = 1$:

$$R_2^{-1}\frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i(0.x_1 1)}|1\rangle) = \frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i(0.x_1 0)}|1\rangle)$$

Phase estimation

$$R_2^{-1} = \begin{bmatrix} 1 & 0 \\ 0 & e^{-2\pi i(0.01)} \end{bmatrix}$$

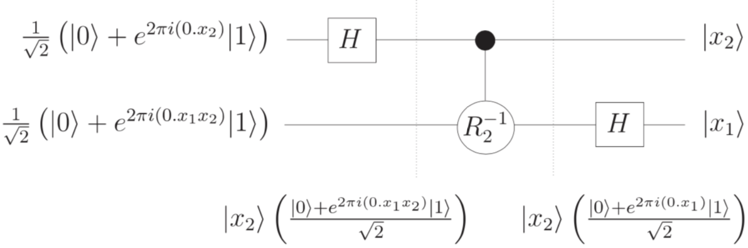
R_2 *only* rotates the phase on $|1\rangle$.
Action when $x_2 = 1$:

$$R_2^{-1} \frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i(0.x_1 1)}|1\rangle) = \frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i(0.x_1 0)}|1\rangle)$$

So we want to perform R_2^{-1} only when $x_2 = 1$! So we use c-R_2^{-1} instead.

Phase estimation

We are finished!



Phase estimation technicalities

This analysis is really incomplete:

- We only saw the action $\omega = 0.x_1 x_2$. What if ω is actually like $2/3$, and cannot be written as $x/2^n$?

Phase estimation technicalities

This analysis is really incomplete:

- We only saw the action $\omega = 0.x_1x_2$. What if ω is actually like $2/3$, and cannot be written as $x/2^n$?
- Circuit gets more and more complicated for more qubits, but follows the same idea of 'pulling' then repeatedly eliminating.

Phase estimation backwards: QFT

When $\omega \approx 0.x_1x_2, \ldots x_n$, phase estimation now gives us

$$\frac{1}{\sqrt{2}^n} \sum_{i=0}^{2^n-1} e^{2\pi i \omega y} |y\rangle \mapsto |x_1x_2 \ldots x_n\rangle$$

With high probability.

Phase estimation backwards: QFT

When $\omega \approx 0.x_1 x_2, \dots x_n$, phase estimation now gives us

$$\frac{1}{\sqrt{2}^n} \sum_{i=0}^{2^n-1} e^{2\pi i \omega y} |y\rangle \mapsto |x_1 x_2 \dots x_n\rangle$$

With high probability.
If we just reverse the circuit, we get

$$|x_1 x_2 \dots x_n\rangle \mapsto \frac{1}{\sqrt{2}^n} \sum_{i=0}^{2^n-1} e^{2\pi i \omega y} |y\rangle$$

Phase estimation backwards: QFT

When $\omega \approx 0.x_1 x_2, \ldots x_n$, phase estimation now gives us

$$\frac{1}{\sqrt{2}^n} \sum_{i=0}^{2^n-1} e^{2\pi i \omega y} \left| y \right\rangle \mapsto \left| x_1 x_2 \ldots x_n \right\rangle$$

With high probability.
If we just reverse the circuit, we get

$$\left| x_1 x_2 \ldots x_n \right\rangle \mapsto \frac{1}{\sqrt{2}^n} \sum_{i=0}^{2^n-1} e^{2\pi i \omega y} \left| y \right\rangle$$

This replicates the behavior of the discrete fourier transform in a quantum implementation, hence QFT. Phase estimation is called "QFT^{-1}".

Back to factoring

FACTOR
Input: $N = pq$
Output: A factor of N, which is either p or q.

Back to factoring

FACTOR
Input: $N = pq$
Output: A factor of N, which is either p or q.

It turns out this reduces (probabilistically) to a problem called ORDER-FIND (if we have a fast algorithm for ORDER-FIND, we can use it as a black box to implement FACTOR)

Back to factoring

FACTOR
Input: $N = pq$
Output: A factor of N, which is either p or q.

It turns out this reduces (probabilistically) to a problem called ORDER-FIND (if we have a fast algorithm for ORDER-FIND, we can use it as a black box to implement FACTOR)

The idea of reductions is very common in computer science, we write FACTOR $\rightarrow$ ORDER-FIND

Order finding

ORDER-FIND
Input: a, N with $\gcd(a, N) = 1$.
Output: Minimum r such that $a^r = 1 \pmod{N}$

Order finding

ORDER-FIND
Input: a, N with $\gcd(a, N) = 1$.
Output: Minimum r such that $a^r = 1 \pmod N$

How come FACTOR $\rightarrow$ ORDER-FIND?

FACTOR → ORDER-FIND?

We can pick elements $a \in \{0, \ldots N-1\}$ randomly.

Factor → Order-find?

We can pick elements $a \in \{0, \ldots N-1\}$ randomly.
Checking that $\gcd(a, N) = 1$ is really quick to do classically (Euclid GCD algorithm).

Factor → Order-find?

We can pick elements $a \in \{0, \ldots N-1\}$ randomly.
Checking that $\gcd(a, N) = 1$ is really quick to do classically (Euclid GCD algorithm).
If the order of a is even, then we've lucked out!

Factor → Order-find?

We can pick elements $a \in \{0, \ldots N-1\}$ randomly.
Checking that $\gcd(a, N) = 1$ is really quick to do classically (Euclid GCD algorithm).
If the order of a is even, then we've lucked out!

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

Factor $\to$ Order-find?

We can pick elements $a \in \{0, \ldots N-1\}$ randomly.
Checking that $\gcd(a, N) = 1$ is really quick to do classically (Euclid GCD algorithm).
If the order of a is even, then we've lucked out!

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

We've just factorized N! $a^{r/2} + 1$ and $a^{r/2} - 1$ are almost always nontrivial (meaning not 1 and N) factors.

Order-find $\rightarrow$ Eigenvalue-estimate

How can we get r, the order, from a quantum algorithm?

Order-find → Eigenvalue-estimate

How can we get r, the order, from a quantum algorithm? Consider

$$U_a\colon\ |s\rangle \mapsto |sa\rangle \pmod{N}$$

Order-find → Eigenvalue-estimate

How can we get r, the order, from a quantum algorithm?
Consider

$$U_a \colon |s\rangle \mapsto |sa\rangle \pmod N$$

$$U_a^r \colon |s\rangle \mapsto |sa^r\rangle \pmod N = |s\rangle$$

Order-find → Eigenvalue-estimate

How can we get r, the order, from a quantum algorithm?
Consider

$$U_a\colon\ |s\rangle \mapsto |sa\rangle \pmod{N}$$

$$U_a^r\colon\ |s\rangle \mapsto |sa^r\rangle \pmod{N} = |s\rangle$$

So $U_a^r = I$!
This means its eigenvalues are all rth roots of unity, i.e. complex numbers of the form

$$e^{2\pi ik/r}, k \in \{0, \dots, r-1\}$$

Order-find → Eigenvalue-estimate

For an eigenvector of U_a, $|\psi\rangle$

$$U_a |\psi\rangle = e^{2\pi ik/r} |\psi\rangle$$

Order-find $\rightarrow$ Eigenvalue-estimate

For an eigenvector of U_a, $|\psi\rangle$

$$U_a |\psi\rangle = e^{2\pi i k/r} |\psi\rangle$$

Modulo some details (k, ψ), we already have a way to estimate $\omega = k/r$ from phases!!

Eigenvalue estimation

EIGENVALUE-ESTIMATE
Input: A unitary operator U implemented in quantum gates, and an eigenvector $|\psi\rangle$
Output: ω such that $U|\psi\rangle = e^{2\pi i\omega}|\psi\rangle$

Eigenvalue estimation

EIGENVALUE-ESTIMATE
Input: A unitary operator U implemented in quantum gates, and an eigenvector $|\psi\rangle$
Output: ω such that $U|\psi\rangle = e^{2\pi i\omega}|\psi\rangle$
(as before, estimated as $\omega = \frac{x}{2^n}$ with $x \in \{0, \ldots, 2^n - 1\}$

Eigenvalue estimation

EIGENVALUE-ESTIMATE
Input: A unitary operator U implemented in quantum gates, and an eigenvector $|\psi\rangle$
Output: ω such that $U|\psi\rangle = e^{2\pi i\omega}|\psi\rangle$
(as before, estimated as $\omega = \frac{x}{2^n}$ with $x \in \{0, \ldots, 2^n - 1\}$

Idea: Apply U repeatedly, so we get a quantum state where we can estimate ω from our phase estimation algorithm

Eigenvalue estimation implementation

$$U|\psi\rangle = e^{2\pi i 0.x_1 \ldots x_n}|\psi\rangle$$

Eigenvalue estimation implementation

$$U\left|\psi\right\rangle = e^{2\pi i 0.x_1 \ldots x_n}\left|\psi\right\rangle$$

$$U^2\left|\psi\right\rangle = (e^{2\pi i 0.x_1 \ldots x_n})^2\left|\psi\right\rangle = e^{2\pi i x_1 . x_2 \ldots x_{n-1}}\left|\psi\right\rangle = e^{2\pi i 0.x_2 \ldots x_{n-1}}\left|\psi\right\rangle$$

Eigenvalue estimation implementation

$$U|\psi\rangle = e^{2\pi i 0.x_1 \dots x_n}|\psi\rangle$$

$$U^2|\psi\rangle = (e^{2\pi i 0.x_1 \dots x_n})^2|\psi\rangle = e^{2\pi i x_1.x_2 \dots x_{n-1}}|\psi\rangle = e^{2\pi i 0.x_2 \dots x_{n-1}}|\psi\rangle$$

$$U^{2^j}|\psi\rangle = e^{2\pi i 0.x_j \dots x_{n-j}}|\psi\rangle$$

Eigenvalue estimation implementation

$$U|\psi\rangle = e^{2\pi i 0.x_1 \ldots x_n}|\psi\rangle$$

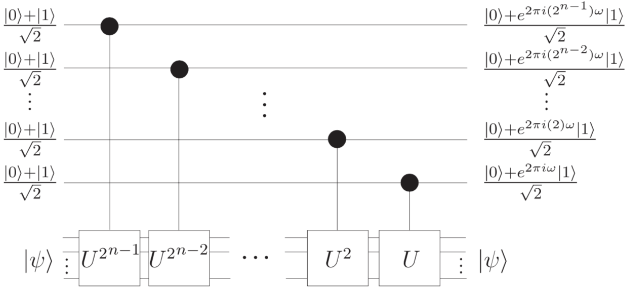
$$U^2|\psi\rangle = (e^{2\pi i 0.x_1 \ldots x_n})^2|\psi\rangle = e^{2\pi i x_1.x_2 \ldots x_{n-1}}|\psi\rangle = e^{2\pi i 0.x_2 \ldots x_{n-1}}|\psi\rangle$$

$$U^{2^j}|\psi\rangle = e^{2\pi i 0.x_j \ldots x_{n-j}}|\psi\rangle$$

So we can use c-U^{2^j} to get the individual qu-bits in the state we want for eigenvalue estimation.

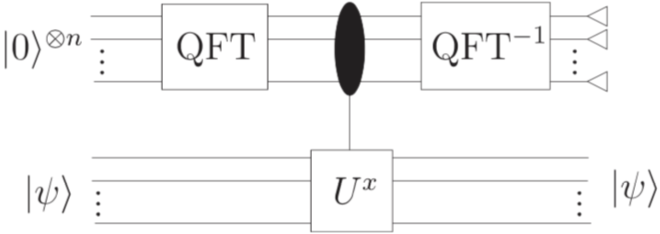
Eigenvalue estimation implementation

So here is our way to 'set up' the states for estimation:



Eigenvalue estimation implementation

It turns out QFT sets up 0 states to $\frac{|0\rangle+|1\rangle}{2}$, so here is our entire diagram (including actually doing phase estimation):



Shor's 'entire' algorithm

We have talked about a way to implement Shor's algorithm (ignoring crucial details like time complexity and correctness):

$$\textsc{Factor} \to \textsc{Order-find} \to$$

$$\textsc{Eigenvalue-estimate} \to \textsc{Phase-estimate}$$

Implications

Factoring numbers quickly breaks security assumptions of RSA.

Implications

Factoring numbers quickly breaks security assumptions of RSA.

This isn't a huge deal yet. The largest number factored with Shor's algorithm is 21, RSA numbers are on the order of 2^{1024}.

Implications

Factoring numbers quickly breaks security assumptions of RSA.

This isn't a huge deal yet. The largest number factored with Shor's algorithm is 21, RSA numbers are on the order of 2^{1024}.

But be careful what you tell people about in 20 years.

End

Thank you!

Diagrams and much of material from 'An Introduction to Quantum Computing', Kaye, Laflamme, Mosca