Grover Search and Its Cryptographic Applications

Henry Corrigan-Gibbs
Qualifying Exam Talk

21 November 2016

Quantum Computing and Crypto

Large-scale quantum computers could exist in our lifetimes.

Quantum Computing and Crypto

Large-scale quantum computers could exist in our lifetimes.

Quantum computers can break today's crypto primitives!

Quantum Computing and Crypto

Large-scale quantum computers could exist in our lifetimes.

Quantum computers can break today's crypto primitives!

	Examples	**Outcome**
Public-key	RSA, DH, ECDH	Broken (Shor)

Quantum Computing and Crypto

Large-scale quantum computers could exist in our lifetimes.

Quantum computers can break today's crypto primitives!

	Examples	**Outcome**
Public-key	RSA, DH, ECDH	Broken (Shor)
Modes of operation	GCM, CBC-MAC	Broken* (Simon)

Quantum Computing and Crypto

Large-scale quantum computers could exist in our lifetimes.

Quantum computers can break today's crypto primitives!

	Examples	**Outcome**
Public-key	RSA, DH, ECDH	Broken (Shor)
Modes of operation	GCM, CBC-MAC	Broken* (Simon)
Block ciphers	AES, DES	Attacks improve (Grover)

Quantum Computing and Crypto

Large-scale quantum computers could exist in our lifetimes.

Quantum computers can break today's crypto primitives!

	Examples	**Outcome**
Public-key	RSA, DH, ECDH	Broken (Shor)
Modes of operation	GCM, CBC-MAC	Broken* (Simon)
Block ciphers	AES, DES	Attacks improve (Grover)
Hash functions	SHA2	Attacks improve* (Grover)

Quantum Computing and Crypto

Large-scale quantum computers could exist in our lifetimes.

Quantum computers can break today's crypto primitives!

	Examples	**Outcome**
Public-key	RSA, DH, ECDH	Broken (Shor)
Modes of operation	GCM, CBC-MAC	Broken* (Simon)
Block ciphers	AES, DES	Attacks improve (Grover)
Hash functions	SHA2	Attacks improve* (Grover)
Password hashing	PBKDF2, scrypt	Broken* (Grover)

Quantum Computing and Crypto

Large-scale quantum computers could exist in our lifetimes.

Quantum computers can break today's crypto primitives!

	Examples	**Outcome**
Public-key	RSA, DH, ECDH	Broken (Shor)
Modes of operation	GCM, CBC-MAC	Broken* (Simon)
Block ciphers	AES, DES	Attacks improve (Grover)
Hash functions	SHA2	Attacks improve* (Grover)
Password hashing	PBKDF2, scrypt	Broken* (Grover)

$\Rightarrow$ To design good post-quantum cryptosystems, we need to understand post-quantum cryptanalysis.

Quantum Computing and Crypto

Large-scale quantum computers could exist in our lifetimes.

Quantum computers can break today's crypto primitives!

	Examples	**Outcome**
Public-key	RSA, DH, ECDH	Broken (Shor)
Modes of operation	GCM, CBC-MAC	Broken* (Simon)
Block ciphers	AES, DES	Attacks improve (Grover)
Hash functions	SHA2	Attacks improve* (Grover)
Password hashing	PBKDF2, scrypt	Broken* (Grover)

This talk

⇒ To design good post-quantum cryptosystems, we need to understand post-quantum cryptanalysis.

Quantum Computing and Crypto

Large-scale quantum computers could exist in our lifetimes.

Quantum computers can break today's crypto primitives!

	Examples	**Outcome**
Public-key	RSA, DH, ECDH	Broken (Shor)
Modes of operation	GCM, CBC-MAC	Broken* (Simon)
Block ciphers	AES, DES	Attacks impro
Hash functions	SHA2	Attacks improve
Password hashing	PBKDF2, scrypt	Broken* (Grover)

You heard it here first!

⇒ To design good post-quantum cryptosystems, we need to understand post-quantum cryptanalysis.

Quantum Computing and Crypto

Large-scale quantum computers could exist in our lifetimes.

Quantum computers can break today's crypto primitives!

	Examples	**Outcome**
Public-key	RSA, DH, ECDH	Broken (Shor)
Modes of operation	GCM, CBC-MAC	Broken* (Simon)
Block ciphers	AES, DES	Attacks improve (Grover)
Hash functions	SHA2	Attacks improve* (Grover)
Password hashing	PBKDF2, scrypt	Broken* (Grover)

$\Rightarrow$ To design good post-quantum cryptosystems, we need to understand post-quantum cryptanalysis.

Overview

Warm up: Probabilistic Computation

(Following the treatment of Arora and Barak.)

By analogy to probabilistic computation. . .

Warm up: Probabilistic Computation

(Following the treatment of Arora and Barak.)

By analogy to probabilistic computation...

An example computation.

1. Initialize a two-bit register with input.
2. Swap the two bits with probability $1/2$.
3. Output the register state.

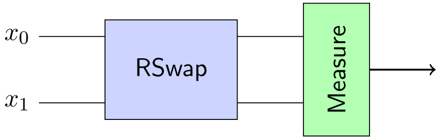
Warm up: Probabilistic Computation

(Following the treatment of Arora and Barak.)

By analogy to probabilistic computation. . .

An example computation.

1. Initialize a two-bit register with input.
2. Swap the two bits with probability $1/2$.
3. Output the register state.



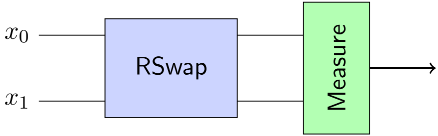
Warm up: Probabilistic Computation

(Following the treatment of Arora and Barak.)

By analogy to probabilistic computation. . .

An example computation.

1. Initialize a two-bit register with input.
2. Swap the two bits with probability $1/2$.
3. Output the register state.



Input	$\mapsto$	*Output*
00		00
01		01 or 10
10		10 or 01
11		11

Warm up: State of Probabilistic Machine

- We can describe the distribution over register states (**00**, **01**, **10**, **11**) with a vector in $\mathbb{R}^4$.

Warm up: State of Probabilistic Machine

- We can describe the distribution over register states (**00**, **01**, **10**, **11**) with a vector in $\mathbb{R}^4$.
- Reading the contents of the register gives a sample from this distribution.

Warm up: State of Probabilistic Machine

- We can describe the distribution over register states (**00**, **01**, **10**, **11**) with a vector in $\mathbb{R}^4$.
- Reading the contents of the register gives a sample from this distribution.

$$\mathbb{R}^4 \ni \begin{pmatrix} \alpha_{00} \\ \alpha_{01} \\ \alpha_{10} \\ \alpha_{11} \end{pmatrix} \begin{matrix} \leftarrow \text{Prob. of "}\mathbf{00}\text{"} \\ \leftarrow \text{Prob. of "}\mathbf{01}\text{"} \\ \leftarrow \text{Prob. of "}\mathbf{10}\text{"} \\ \leftarrow \text{Prob. of "}\mathbf{11}\text{"} \end{matrix}$$

Warm up: State of Probabilistic Machine

- We can describe the distribution over register states (**00**, **01**, **10**, **11**) with a vector in $\mathbb{R}^4$.
- Reading the contents of the register gives a sample from this distribution.

$$\mathbb{R}^4 \ni \begin{pmatrix} \alpha_{00} \\ \alpha_{01} \\ \alpha_{10} \\ \alpha_{11} \end{pmatrix} \begin{matrix} \leftarrow \text{Prob. of "00"} \\ \leftarrow \text{Prob. of "01"} \\ \leftarrow \text{Prob. of "10"} \\ \leftarrow \text{Prob. of "11"} \end{matrix}$$

Every possible state is a linear combination of basis states:

$$|00\rangle = \begin{pmatrix} \mathbf{1} \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad |01\rangle = \begin{pmatrix} 0 \\ \mathbf{1} \\ 0 \\ 0 \end{pmatrix}, \quad |10\rangle = \begin{pmatrix} 0 \\ 0 \\ \mathbf{1} \\ 0 \end{pmatrix}, \quad |11\rangle = \begin{pmatrix} 0 \\ 0 \\ 0 \\ \mathbf{1} \end{pmatrix}$$

N.B. $|0\rangle|1\rangle = |01\rangle$.

Warm up: State of Probabilistic Machine

- We can describe the distribution over register states (**00**, **01**, **10**, **11**) with a vector in $\mathbb{R}^4$.
- Reading the contents of the register gives a sample from this distribution.

$$\mathbb{R}^4 \ni \begin{pmatrix} \alpha_{00} \\ \alpha_{01} \\ \alpha_{10} \\ \alpha_{11} \end{pmatrix} \begin{matrix} \leftarrow \text{Prob. of “}\mathbf{00}\text{”} \\ \leftarrow \text{Prob. of “}\mathbf{01}\text{”} \\ \leftarrow \text{Prob. of “}\mathbf{10}\text{”} \\ \leftarrow \text{Prob. of “}\mathbf{11}\text{”} \end{matrix}$$

Every ... linear combination of basis states:

Dirac's very useful “ket” notation

$$|00\rangle = \begin{pmatrix} \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad |01\rangle = \begin{pmatrix} 0 \\ \mathbf{1} \\ 0 \\ 0 \end{pmatrix}, \quad |10\rangle = \begin{pmatrix} 0 \\ 0 \\ \mathbf{1} \\ 0 \end{pmatrix}, \quad |11\rangle = \begin{pmatrix} 0 \\ 0 \\ 0 \\ \mathbf{1} \end{pmatrix}$$

N.B. $|0\rangle|1\rangle = |01\rangle$.

Warm up: State of Probabilistic Machine

- We can describe the distribution over register states (**00**, **01**, **10**, **11**) with a vector in $\mathbb{R}^4$.
- Reading the contents of the register gives a sample from this distribution.

$$\mathbb{R}^4 \ni \begin{pmatrix} \alpha_{00} \\ \alpha_{01} \\ \alpha_{10} \\ \alpha_{11} \end{pmatrix} \begin{matrix} \leftarrow \text{Prob. of "}\mathbf{00}\text{"} \\ \leftarrow \text{Prob. of "}\mathbf{01}\text{"} \\ \leftarrow \text{Prob. of "}\mathbf{10}\text{"} \\ \leftarrow \text{Prob. of "}\mathbf{11}\text{"} \end{matrix}$$

Every possible state is a linear combination of basis states:

$$|00\rangle = \begin{pmatrix} \mathbf{1} \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad |01\rangle = \begin{pmatrix} 0 \\ \mathbf{1} \\ 0 \\ 0 \end{pmatrix}, \quad |10\rangle = \begin{pmatrix} 0 \\ 0 \\ \mathbf{1} \\ 0 \end{pmatrix}, \quad |11\rangle = \begin{pmatrix} 0 \\ 0 \\ 0 \\ \mathbf{1} \end{pmatrix}$$

N.B. $|0\rangle|1\rangle = |01\rangle$.

Warm up: Probabilistic Operations

We can use *stochastic matrix* to describe the action of the swap gate on the register state.

Warm up: Probabilistic Operations

We can use *stochastic matrix* to describe the action of the swap gate on the register state.

$$S = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1/2 & 1/2 & 0 \\ 0 & 1/2 & 1/2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Warm up: Probabilistic Operations

We can use *stochastic matrix* to describe the action of the swap gate on the register state.

$$S = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1/2 & 1/2 & 0 \\ 0 & 1/2 & 1/2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$S|\mathbf{00}\rangle \mapsto |\mathbf{00}\rangle \qquad S|\mathbf{10}\rangle \mapsto \frac{1}{2}(|\mathbf{01}\rangle + |\mathbf{10}\rangle)$$

$$S|\mathbf{01}\rangle \mapsto \frac{1}{2}(|\mathbf{01}\rangle + |\mathbf{10}\rangle) \qquad S|\mathbf{11}\rangle \mapsto |\mathbf{11}\rangle$$

Warm up: Probabilistic Operations

We can use *stochastic matrix* to describe the action of the swap gate on the register state.

$$S = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1/2 & 1/2 & 0 \\ 0 & 1/2 & 1/2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$S|\mathbf{00}\rangle \mapsto |\mathbf{00}\rangle \qquad S|\mathbf{10}\rangle \mapsto \frac{1}{2}(|\mathbf{01}\rangle + |\mathbf{10}\rangle)$$

$$S|\mathbf{01}\rangle \mapsto \frac{1}{2}(|\mathbf{01}\rangle + |\mathbf{10}\rangle) \qquad S|\mathbf{11}\rangle \mapsto |\mathbf{11}\rangle$$

$\Rightarrow$ Computation is just a matrix-vector product.

Probabilistic Computation

Register state: a vector in $\mathbb{R}^{2^n}$.

Probabilistic Computation

Register state: a vector in $\mathbb{R}^{2^n}$.

Probabilistic Computation

1. **Initialize** the register to $|x\rangle$, on input $x \in \{0,1\}^n$.

Probabilistic Computation

Register state: a vector in $\mathbb{R}^{2^n}$.

Probabilistic Computation

1. **Initialize** the register to $|x\rangle$, on input $x \in \{0,1\}^n$.
2. **Run** the computation by computing a matrix-vector product $F_T \cdots F_3 F_2 F_1 |x\rangle$ (i.e., apply the circuit to the register).

Probabilistic Computation

Register state: a vector in $\mathbb{R}^{2^n}$.

Probabilistic Computation

1. **Initialize** the register to $|x\rangle$, on input $x \in \{0,1\}^n$.
2. **Run** the computation by computing a matrix-vector product $F_T \cdots F_3 F_2 F_1 |x\rangle$ (i.e., apply the circuit to the register).
3. **Measure** the register.

Probabilistic Computation

Register state: a vector in $\mathbb{R}^{2^n}$.

Probabilistic Computation

1. **Initialize** the register to $|x\rangle$, on input $x \in \{0,1\}^n$.
2. **Run** the computation by computing a matrix-vector product $F_T \cdots F_3 F_2 F_1 |x\rangle$ (i.e., apply the circuit to the register).
3. **Measure** the register.

If the output of the computation is $\sum_y \alpha_y |y\rangle$, we will measure y with probability α_y.

Probabilistic Computation

Register state: a vector in $\mathbb{R}^{2^n}$.

Probabilistic Computation

1. **Initialize** the register to $|x\rangle$, on input $x \in \{0,1\}^n$.
2. **Run** the computation by computing a matrix-vector product $F_T \cdots F_3 F_2 F_1 |x\rangle$ (i.e., apply the circuit to the register).
3. **Measure** the register.

If the output of the computation is $\sum_y \alpha_y |y\rangle$, we will measure y with probability α_y.

We require that F_is:

Probabilistic Computation

Register state: a vector in $\mathbb{R}^{2^n}$.

Probabilistic Computation

1. **Initialize** the register to $|x\rangle$, on input $x \in \{0,1\}^n$.
2. **Run** the computation by computing a matrix-vector product $F_T \cdots F_3 F_2 F_1 |x\rangle$ (i.e., apply the circuit to the register).
3. **Measure** the register.

If the output of the computation is $\sum_y \alpha_y |y\rangle$, we will measure y with probability α_y.

We require that F_is:

- come from a fixed set of universal gates (AND, OR, etc.),

Probabilistic Computation

Register state: a vector in $\mathbb{R}^{2^n}$.

Probabilistic Computation

1. **Initialize** the register to $|x\rangle$, on input $x \in \{0,1\}^n$.
2. **Run** the computation by computing a matrix-vector product $F_T \cdots F_3 F_2 F_1 |x\rangle$ (i.e., apply the circuit to the register).
3. **Measure** the register.

If the output of the computation is $\sum_y \alpha_y |y\rangle$, we will measure y with probability α_y.

We require that F_is:

- come from a fixed set of universal gates (AND, OR, etc.),
- preserve the L_1 norm (i.e., are stochastic matrices).

Probabilistic Computation

Register state: a vector in $\mathbb{R}^{2^n}$.

Probabilistic Computation

1. **Initialize** the register to $|x\rangle$, on input $x \in \{0,1\}^n$.
2. **Run** the computation by computing a matrix-vector product $F_T \cdots F_3 F_2 F_1 |x\rangle$ (i.e., apply the circuit to the register).
3. **Measure** the register.

If the output of the computation is $\sum_y \alpha_y |y\rangle$, we will measure y with probability α_y.

We require that F_is:

- come from a fixed set of universal gates (AND,
- preserve the L_1 norm (i.e., are stochastic matrices).

Probabilities sum to one.

Probabilistic Computation

Register state: a vector in $\mathbb{R}^{2^n}$.

Probabilistic Computation

1. **Initialize** the register to $|x\rangle$, on input $x \in \{0,1\}^n$.
2. **Run** the computation by computing a matrix-vector product $F_T \cdots F_3 F_2 F_1 |x\rangle$ (i.e., apply the circuit to the register).
3. **Measure** the register.

If the output of the computation is $\sum_y \alpha_y |y\rangle$, we will measure y with probability α_y.

We require that F_is:

- come from a fixed set of universal gates (AND, OR, etc.),
- preserve the L_1 norm (i.e., are stochastic matrices).

Quantum Computation

Register state: a vector in $\mathbb{C}^{2^n}$. (A "superposition")

Quantum Computation

1. **Initialize** the register to $|x\rangle$, on input $x \in \{0,1\}^n$.
2. **Run** the computation by computing a matrix-vector product $F_T \cdots F_3 F_2 F_1 |x\rangle$ (i.e., apply the circuit to the register).
3. **Measure** the register.

If the output of the computation is $\sum_y \alpha_y |y\rangle$, we will measure y with probability $|\alpha_y|^2$, where α_y is an "amplitude."

We require that the F_is:

- come from a fixed set of universal gates (H, T, etc.),
- preserve the L_2 norm (i.e., are unitary matrices).

Quantum Computation

Register state: a vector in $\mathbb{C}^{2^n}$. (A "superposition")

Quantum Computation

1. **Initialize** the register to $|x\rangle$, on input $x \in \{0,1\}^n$.
2. **Run** the computation by computing a matrix-vector product $F_T \cdots F_3 F_2 F_1 |x\rangle$ (i.e., apply the circuit to the register).
3. **Measure** the register.

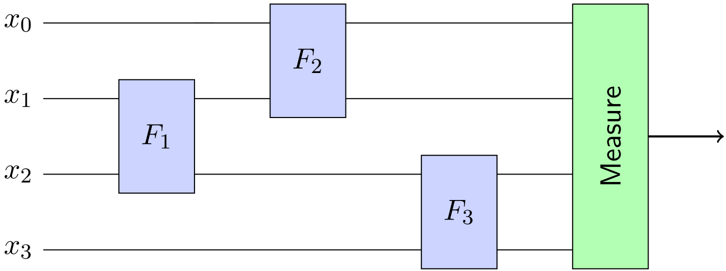
If the output of the computation is $\sum_y \alpha_y |y\rangle$, we will measure y with probability $|\alpha_y|^2$, where α_y is an "amplitude."

We require that the F_is:

- come from a fixed set of universal gates (H,
- preserve the L_2 norm (i.e., are unitary matrices).

Probabilities sum to one.

Example: Quantum Circuit



Observations about QC

Observations about QC

1. Gates must represent unitary transformations ($UU^\dagger = I$), so all computation must be **reversible**.

Observations about QC

1. Gates must represent unitary transformations ($UU^\dagger = I$), so all computation must be **reversible**.

2. Amplitudes can be **negative**, unlike probabilities.
   – This is the source of QC's apparent power.

Useful Tool: Hadamard Gate

Definition

The *Hadamard gate* H is the quantum analogue of a classical bit-flip:

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

Useful Tool: Hadamard Gate

Definition

The *Hadamard gate* H is the quantum analogue of a classical bit-flip:

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

$H|0\rangle \mapsto \frac{|0\rangle + |1\rangle}{\sqrt{2}}$

Useful Tool: Hadamard Gate

Definition

The *Hadamard gate* H is the quantum analogue of a classical bit-flip:

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

$H|0\rangle \mapsto \frac{|0\rangle + |1\rangle}{\sqrt{2}}$

The operator $H^{\otimes n}$ applies H to each of n qubits.

Useful Tool: Quantum Queries

Fact (Lecerf 1963, Bennett 1973)

If $f : \{0,1\}^n \to \{0,1\}$ *is computable with a* $T(n)$*-size classical circuit, then there is a size-*$O(T(n))$ *quantum circuit that maps:*

$$|x\rangle|y\rangle \quad \mapsto \quad |x\rangle|y \oplus f(x)\rangle,$$

possibly using $O(T(n))$ *extra "work" bits.*

Useful Tool: Quantum Queries

Fact (Lecerf 1963, Bennett 1973)

If $f : \{0,1\}^n \to \{0,1\}$ is computable with a $T(n)$-size classical circuit, then there is a size-$O(T(n))$ quantum circuit that maps:

$$|x\rangle|y\rangle \quad \mapsto \quad |x\rangle|y \oplus f(x)\rangle,$$

possibly using $O(T(n))$ extra "work" bits.

Can make quantum queries to a classical function!

Useful Tool: Quantum Queries

Fact (Lecerf 1963, Bennett 1973)

If $f : \{0,1\}^n \to \{0,1\}$ is computable with a $T(n)$-size classical circuit, then there is a size-$O(T(n))$ quantum circuit that maps:

$$|x\rangle|y\rangle \quad \mapsto \quad |x\rangle|y \oplus f(x)\rangle,$$

possibly using $O(T(n))$ extra "work" bits.

There is also a quantum circuit Q_f of similar size that takes:

$$|x\rangle \quad \mapsto \quad (-1)^{f(x)}|x\rangle.$$

Useful Tool: Quantum Queries

Fact (Lecerf 1963, Bennett 1973)

If $f : \{0,1\}^n \to \{0,1\}$ is computable with a $T(n)$-size classical circuit, then there is a size-$O(T(n))$ quantum circuit that maps:

$$|x\rangle|y\rangle \quad \mapsto \quad |x\rangle|y \oplus f(x)\rangle,$$

possibly using $O(T(n))$ extra "work" bits.

There is also a quantum circuit Q_f of similar size that takes:

$$|x\rangle \quad \mapsto \quad (-1)^{f(x)}|x\rangle.$$

This essentially changes the sign of "good" xs in a superposition.

Overview

Definition (Unstructured Search Problem)

Given *oracle access* to a function $f : [N] \to \{0, 1\}$, find a value $x \in [N]$ such that $f(x) = 1$.

Definition (Unstructured Search Problem)

Given *oracle access* to a function $f : [N] \to \{0, 1\}$, find a value $x \in [N]$ such that $f(x) = 1$.

Many cool applications discussed in a moment.

Definition (Unstructured Search Problem)

Given *oracle access* to a function $f : [N] \to \{0, 1\}$, find a value $x \in [N]$ such that $f(x) = 1$.

Many cool applications discussed in a moment.

A few interesting variants:

Definition (Unstructured Search Problem)

Given *oracle access* to a function $f : [N] \to \{0,1\}$, find a value $x \in [N]$ such that $f(x) = 1$.

Many cool applications discussed in a moment.

A few interesting variants:

Unique solution,

Definition (Unstructured Search Problem)

Given *oracle access* to a function $f : [N] \to \{0,1\}$, find a value $x \in [N]$ such that $f(x) = 1$.

Many cool applications discussed in a moment.

A few interesting variants:

Unique solution, Exactly s solutions,

Definition (Unstructured Search Problem)

Given *oracle access* to a function $f : [N] \to \{0, 1\}$, find a value $x \in [N]$ such that $f(x) = 1$.

Many cool applications discussed in a moment.

A few interesting variants:

Unique solution,   Exactly s solutions,   Unknown $\#$ of solutions.

Definition (Unstructured Search Problem)

Given *oracle access* to a function $f : [N] \to \{0,1\}$, find a value $x \in [N]$ such that $f(x) = 1$.

Many cool applications discussed in a moment.

A few interesting variants:
Unique solution, Exactly s solutions, Unknown # of solutions.

Fact

A classical algorithm for unstructured search that succeeds with constant probability must make $\Omega(N)$ *queries.*

Theorem (Grover 1996)

Theorem (Grover 1996)

There is a **quantum** algorithm for unstructured search that makes $O(\sqrt{N})$ **quantum** queries and succeeds with probability at least $2/3$.

Grover's Algorithm

Let $f : \{0,1\}^n \to \{0,1\}$ and let $N = 2^n$.

Grover's Algorithm

Let $f : \{0,1\}^n \to \{0,1\}$ and let $N = 2^n$.

- **Oracle:** operator Q_f that maps $|x\rangle \mapsto (-1)^{f(x)}|x\rangle$.
- We can define an operator Q_0 that inverts the sign of $|0^n\rangle$.
- $H^{\otimes n}$ is the quantum n-bit flip operator.

Grover's Algorithm

Let $f : \{0,1\}^n \to \{0,1\}$ and let $N = 2^n$.

- **Oracle:** operator Q_f that maps $|x\rangle \mapsto (-1)^{f(x)}|x\rangle$.
- We can define an operator Q_0 that inverts the sign of $|0^n\rangle$.
- $H^{\otimes n}$ is the quantum n-bit flip operator.

The Algorithm.

1. **Initialize** an n-bit register to the state $H^{\otimes n}|0^n\rangle$.
2. **Apply** the following operator $O(\sqrt{N})$ times:

$$G = -H^{\otimes n} Q_0 H^{\otimes n} Q_f.$$

3. **Measure** the state of the register and output it.

Analysis of Grover's Algorithm

(Following expositions of Watrous and Jozsa)

Define:

$A = \{x \mid f(x) = 1\}$ ("awesome strings") with $a = |A|$, and

Analysis of Grover's Algorithm

(Following expositions of Watrous and Jozsa)

Define:

$A = \{x \mid f(x) = 1\}$ ("awesome strings") with $a = |A|$, and
$B = \{x \mid f(x) = 0\}$ ("bad strings"), with $b = |B|$.

Analysis of Grover's Algorithm

(Following expositions of Watrous and Jozsa)

Define:

$A = \{x \mid f(x) = 1\}$ ("awesome strings") with $a = |A|$, and
$B = \{x \mid f(x) = 0\}$ ("bad strings"), with $b = |B|$.

Define:

$|A\rangle = \frac{1}{\sqrt{a}} \sum_{x \in A} |x\rangle$, and
$|B\rangle = \frac{1}{\sqrt{b}} \sum_{x \in B} |x\rangle$.

Analysis of Grover's Algorithm

(Following expositions of Watrous and Jozsa)

Define:

$A = \{x \mid f(x) = 1\}$ ("awesome strings") with $a = |A|$, and
$B = \{x \mid f(x) = 0\}$ ("bad strings"), with $b = |B|$.

Define:

$|A\rangle = \frac{1}{\sqrt{a}} \sum_{x \in A} |x\rangle$, and
$|B\rangle = \frac{1}{\sqrt{b}} \sum_{x \in B} |x\rangle$.

Orthogonal unit vectors

Analysis of Grover's Algorithm

(Following expositions of Watrous and Jozsa)

Define:

$A = \{x \mid f(x) = 1\}$ ("awesome strings") with $a = |A|$, and
$B = \{x \mid f(x) = 0\}$ ("bad strings"), with $b = |B|$.

Define:

$|A\rangle = \frac{1}{\sqrt{a}} \sum_{x \in A} |x\rangle$, and
$|B\rangle = \frac{1}{\sqrt{b}} \sum_{x \in B} |x\rangle$.

Analysis of Grover's Algorithm

(Following expositions of Watrous and Jozsa)

Define:

$A = \{x \mid f(x) = 1\}$ ("awesome strings") with $a = |A|$, and
$B = \{x \mid f(x) = 0\}$ ("bad strings"), with $b = |B|$.

Define:

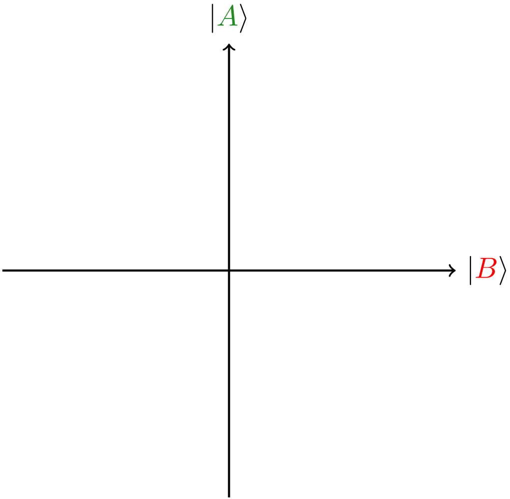
$|A\rangle = \frac{1}{\sqrt{a}} \sum_{x \in A} |x\rangle$, and
$|B\rangle = \frac{1}{\sqrt{b}} \sum_{x \in B} |x\rangle$.

After initialization, the register is in the uniform superposition over strings:

$$H^{\otimes n}|0^n\rangle = |h\rangle = \frac{1}{\sqrt{N}} \sum_x |x\rangle = \underbrace{\sqrt{\frac{a}{N}}|A\rangle}_{\text{Awesome}} + \underbrace{\sqrt{\frac{b}{N}}|B\rangle}_{\text{Bad}}$$

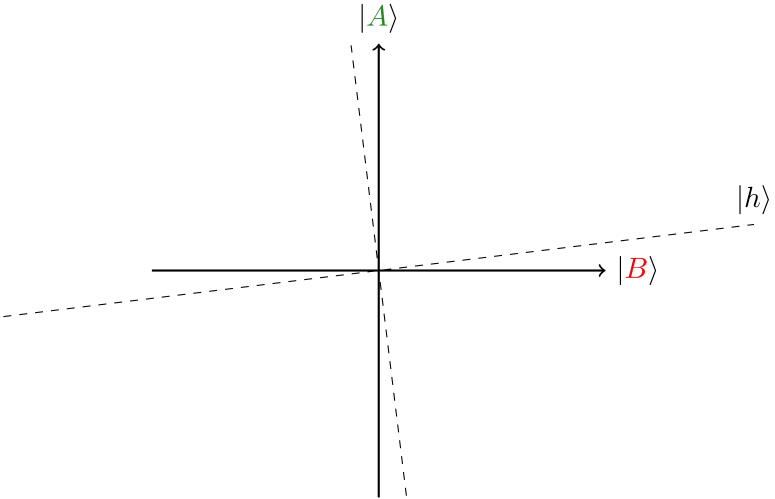
Analysis of Grover's Algorithm

$$G = -H^{\otimes n} Q_0 H^{\otimes n} Q_f$$



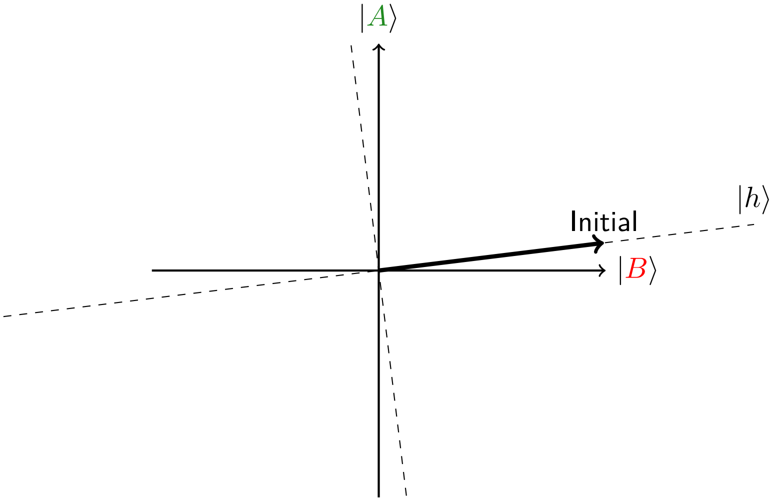
Analysis of Grover's Algorithm

$$G = -H^{\otimes n} Q_0 H^{\otimes n} Q_f$$



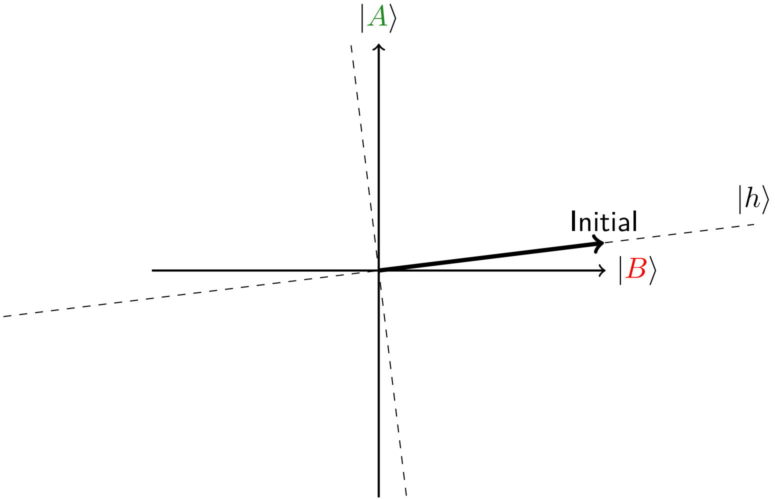
Analysis of Grover's Algorithm

$$G = -H^{\otimes n} Q_0 H^{\otimes n} Q_f$$



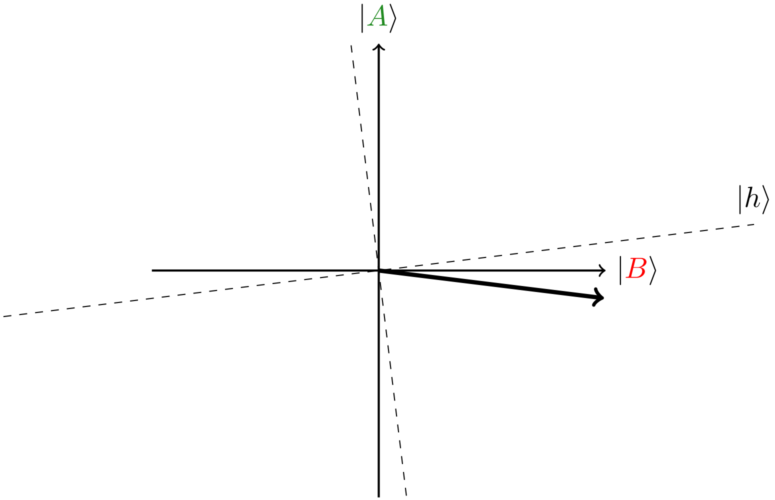
Analysis of Grover's Algorithm

$$G = -H^{\otimes n} Q_0 H^{\otimes n} \boldsymbol{Q}_f$$



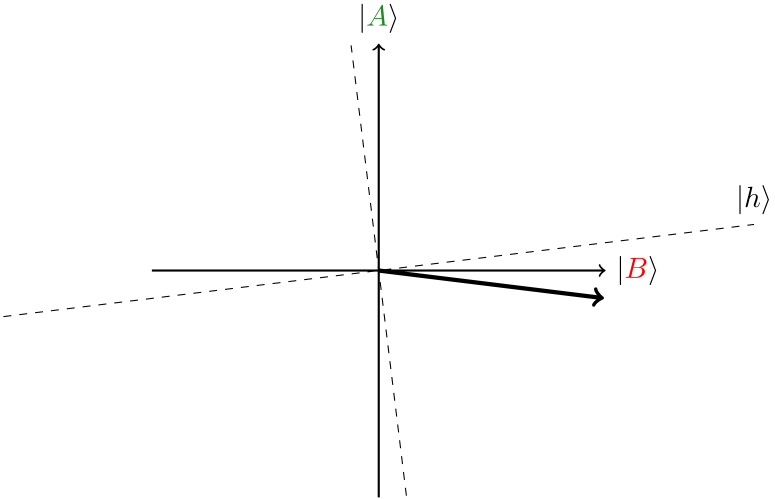
Analysis of Grover's Algorithm

$$G = -H^{\otimes n} Q_0 H^{\otimes n} \boldsymbol{Q}_f$$



Analysis of Grover's Algorithm

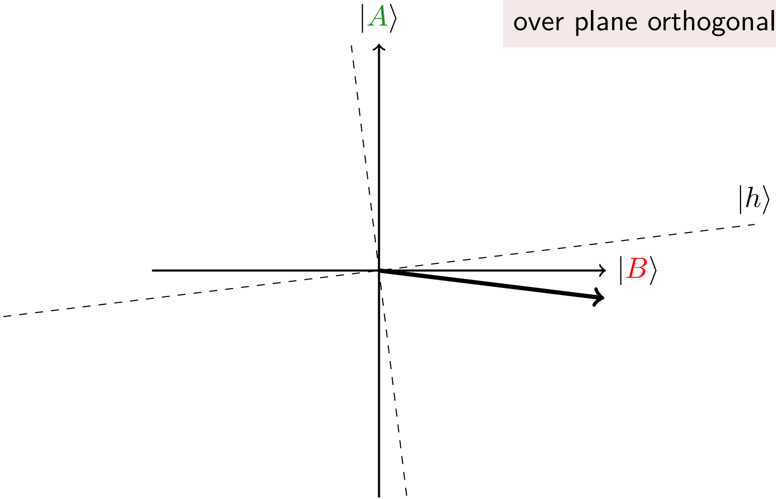
$$G = -\boldsymbol{H}^{\otimes n}\boldsymbol{Q}_0\boldsymbol{H}^{\otimes n}Q_f$$



Analysis of Grover's Algorithm

$$G = -\boldsymbol{H^{\otimes n} Q_0 H^{\otimes n}} Q_f$$

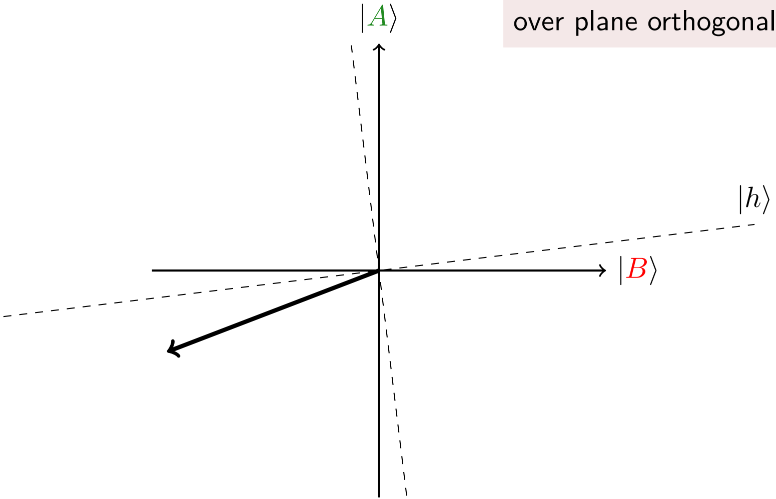
Claim: $H^{\otimes n} Q_0 H^{\otimes n}$ reflects over plane orthogonal to $|h\rangle$.



Analysis of Grover's Algorithm

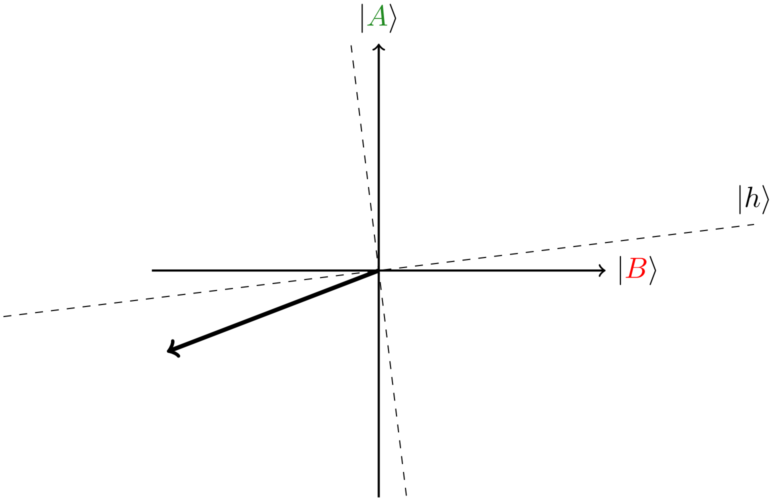
$$G = -H^{\otimes n} Q_0 H^{\otimes n} Q_f$$

Claim: $H^{\otimes n} Q_0 H^{\otimes n}$ reflects over plane orthogonal to $|h\rangle$.



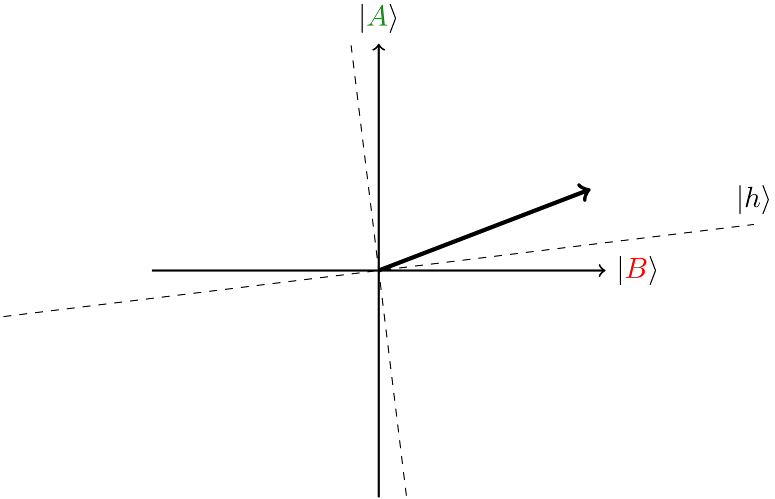
Analysis of Grover's Algorithm

$$G = -H^{\otimes n} Q_0 H^{\otimes n} Q_f$$



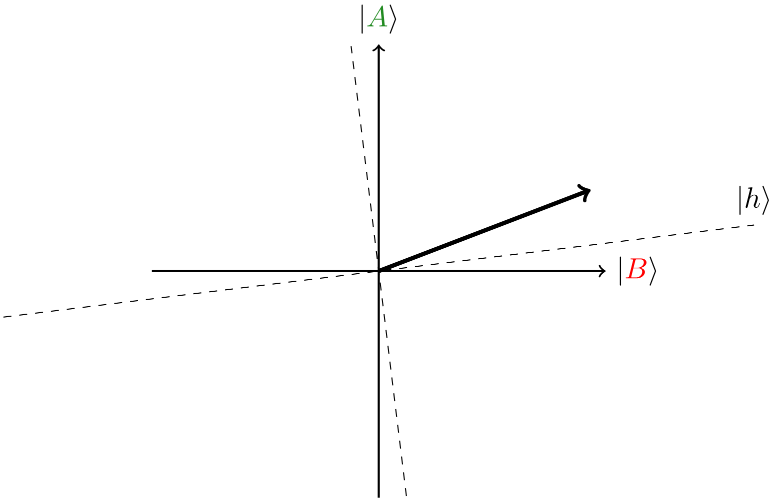
Analysis of Grover's Algorithm

$$G = -H^{\otimes n} Q_0 H^{\otimes n} Q_f$$



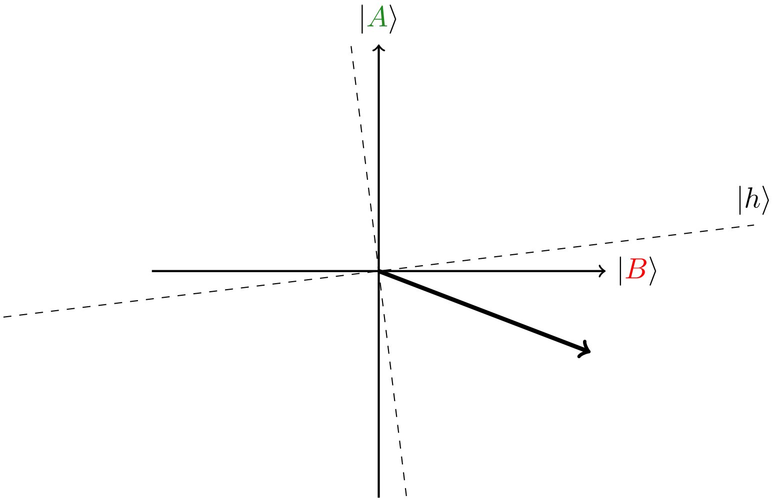
Analysis of Grover's Algorithm

$$G = -H^{\otimes n} Q_0 H^{\otimes n} Q_f$$



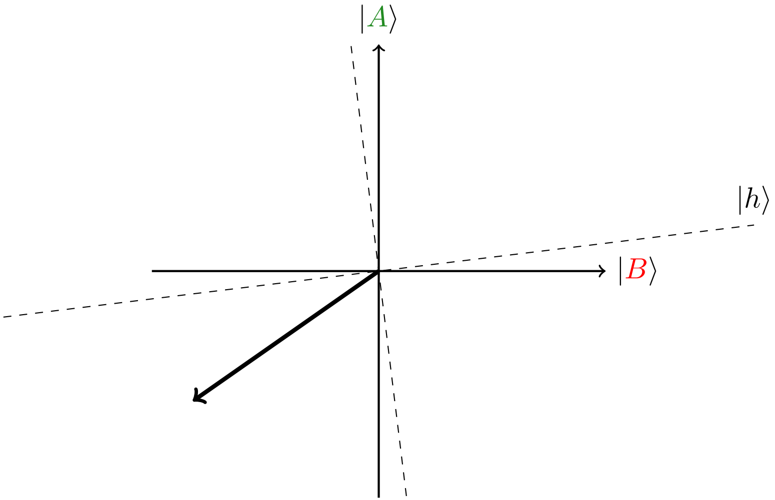
Analysis of Grover's Algorithm

$$G = -H^{\otimes n} Q_0 H^{\otimes n} Q_f$$



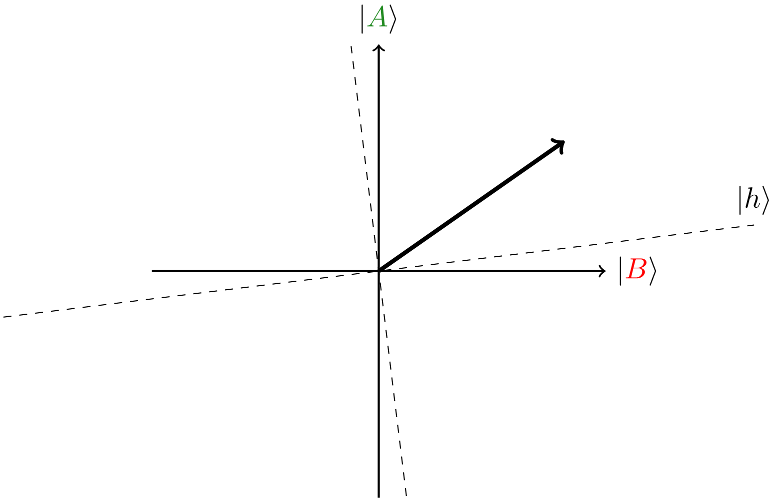
Analysis of Grover's Algorithm

$$G = -H^{\otimes n} Q_0 H^{\otimes n} Q_f$$



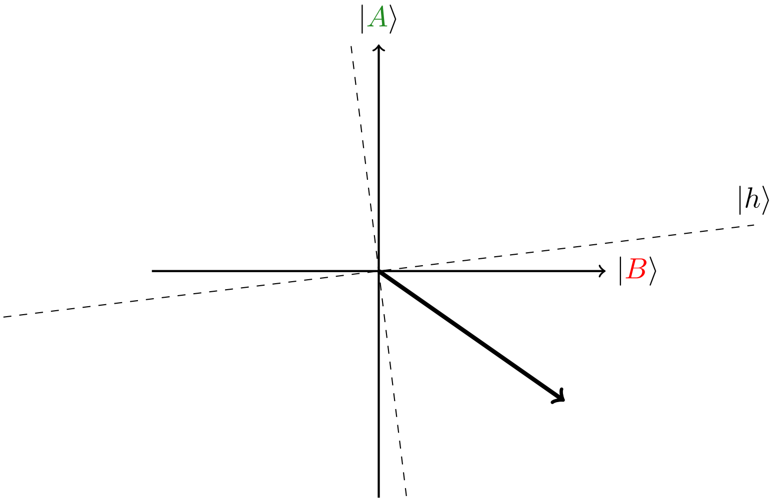
Analysis of Grover's Algorithm

$$G = -H^{\otimes n} Q_0 H^{\otimes n} Q_f$$



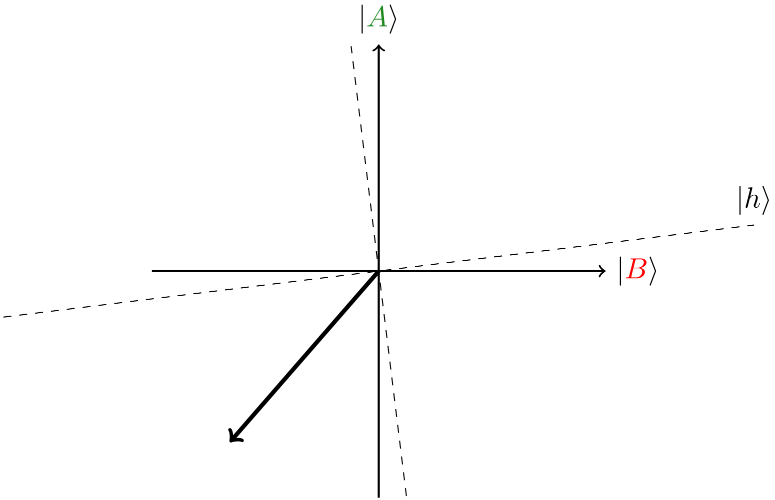
Analysis of Grover's Algorithm

$$G = -H^{\otimes n} Q_0 H^{\otimes n} Q_f$$



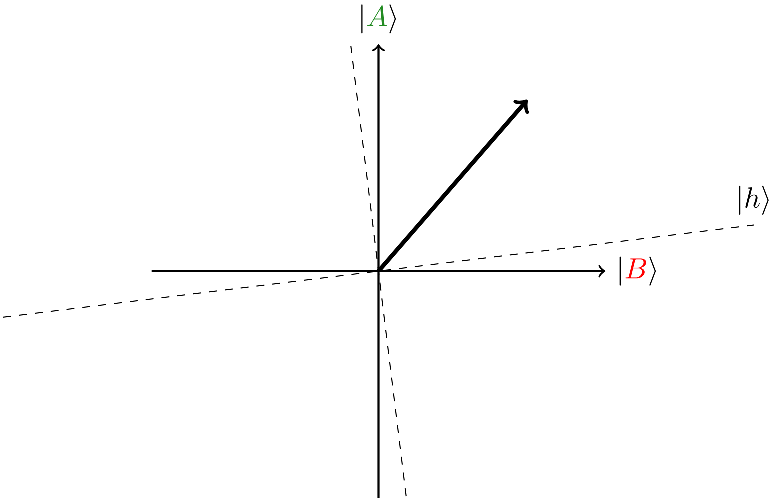
Analysis of Grover's Algorithm

$$G = -H^{\otimes n} Q_0 H^{\otimes n} Q_f$$



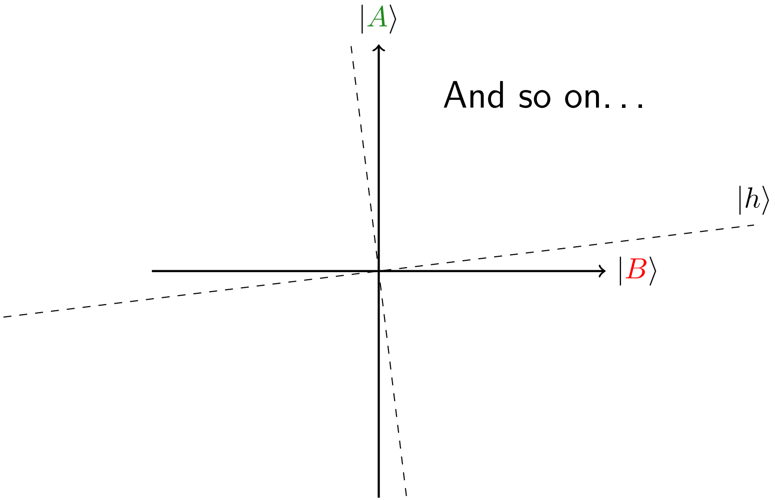
Analysis of Grover's Algorithm

$$G = -H^{\otimes n} Q_0 H^{\otimes n} Q_f$$



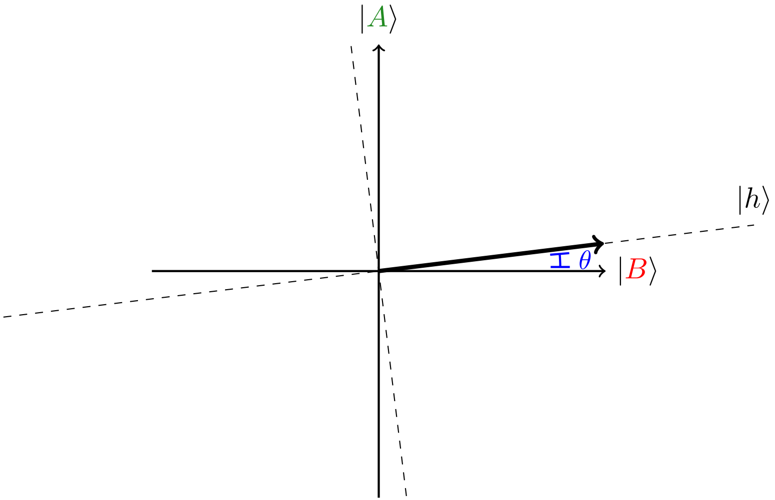
Analysis of Grover's Algorithm

$$G = -H^{\otimes n} Q_0 H^{\otimes n} Q_f$$

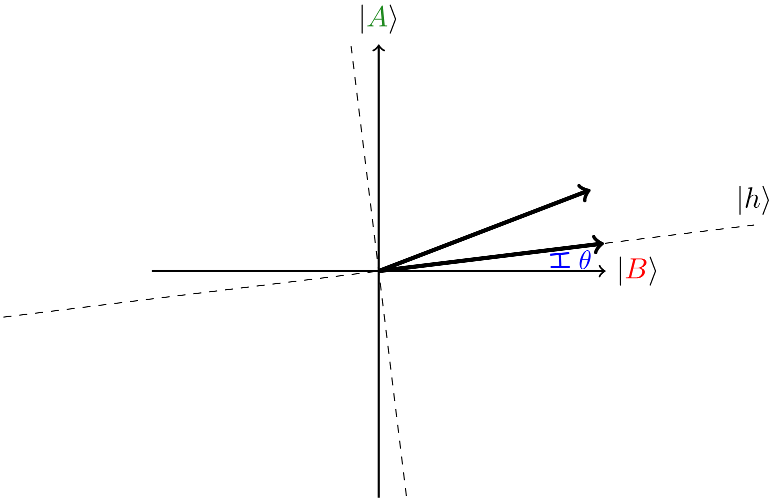


Analysis of Grover's Algorithm

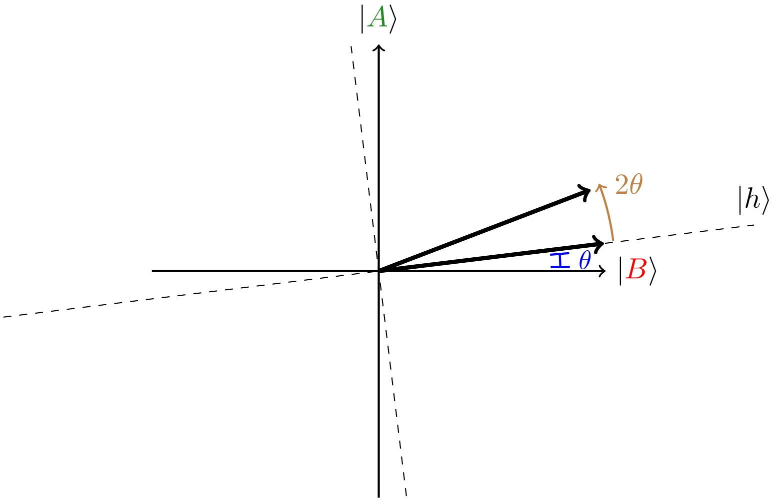
$$G = -H^{\otimes n} Q_0 H^{\otimes n} Q_f$$



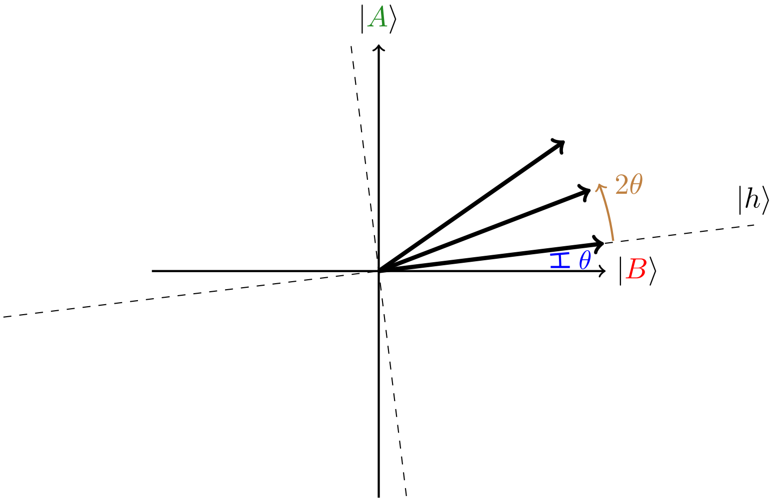
Analysis of Grover's Algorithm



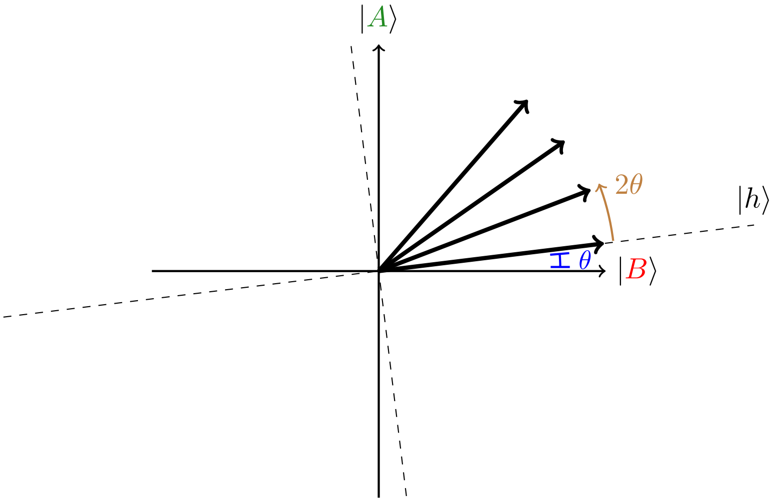
Analysis of Grover's Algorithm



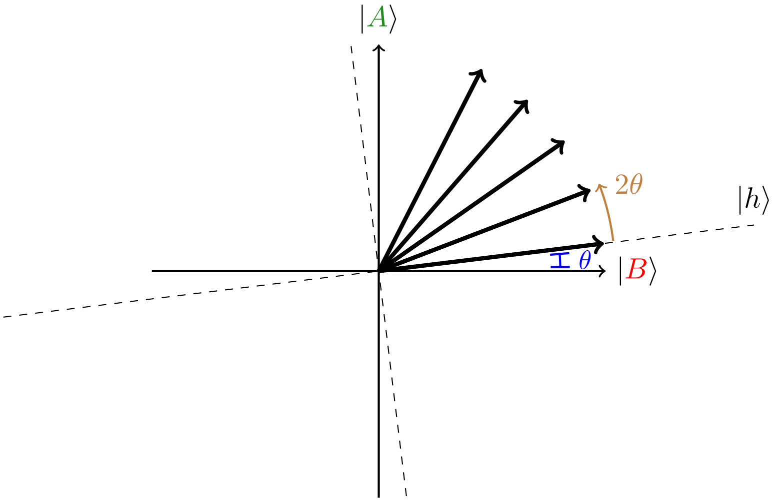
Analysis of Grover's Algorithm



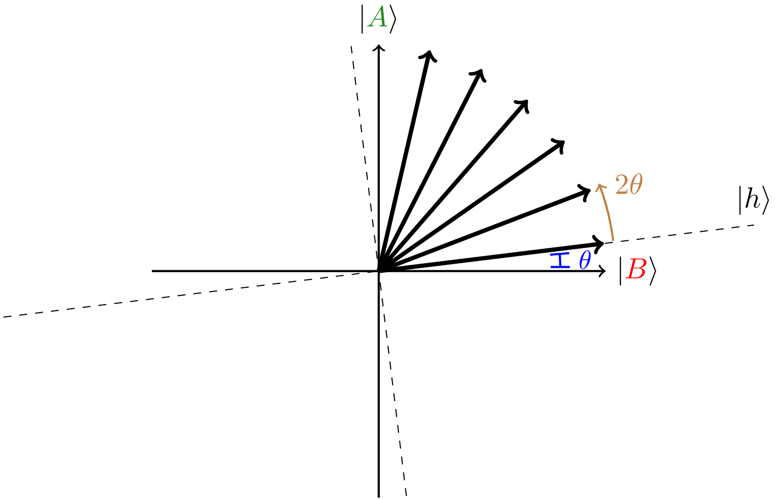
Analysis of Grover's Algorithm



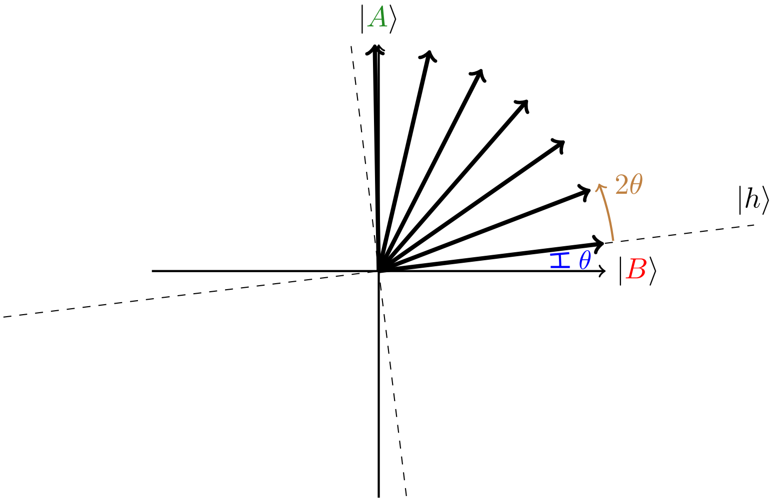
Analysis of Grover's Algorithm



Analysis of Grover's Algorithm

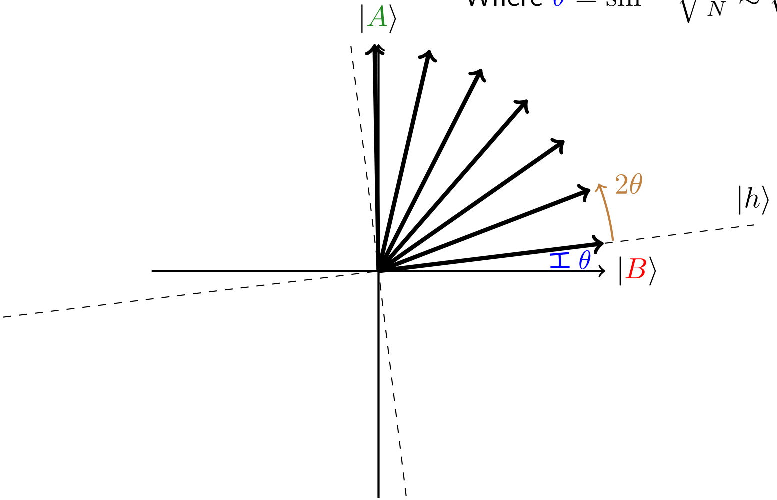


Analysis of Grover's Algorithm



Analysis of Grover's Algorithm

Where $\theta = \sin^{-1}\sqrt{\frac{a}{N}} \approx \sqrt{\frac{a}{N}}$



Analysis of Grover's Algorithm

After t Grover iterations, the angle between the register state and $|B\rangle$ is $\approx 2\theta t$. We want the bad state $|B\rangle$ and the register state to be orthogonal:

$$2\theta t = \frac{\pi}{2}.$$

Analysis of Grover's Algorithm

After t Grover iterations, the angle between the register state and $|B\rangle$ is $\approx 2\theta t$. We want the bad state $|B\rangle$ and the register state to be orthogonal:

$$2\theta t = \frac{\pi}{2}.$$

Num. Solutions	Iterations
1	$\frac{\pi}{4} \cdot \sqrt{N}$
a	$\frac{\pi}{4} \cdot \sqrt{\frac{N}{a}}$
Unknown	$t \leftarrow_R \{1, \ldots, \sqrt{N}\}$

Analysis of Grover's Algorithm

After t Grover iterations, the angle between the register state and $|B\rangle$ is $\approx 2\theta t$. We want the bad state $|B\rangle$ and the register state to be orthogonal:

$$2\theta t = \frac{\pi}{2}.$$

Num. Solutions	Iterations
1	$\frac{\pi}{4} \cdot \sqrt{N}$
a	$\frac{\pi}{4} \cdot \sqrt{\frac{N}{a}}$
Unknown	$t \leftarrow_R \{1, \ldots, \sqrt{N}\}$

One query per iteration $\Rightarrow O(\sqrt{N})$ queries.

Lower Bound

Definition (Decision Grover Problem)

Given oracle access to $f : [N] \to \{0, 1\}$, decide whether there exists an x such that $f(x) = 1$ with probability better than 2/3.

Lower Bound

Definition (Decision Grover Problem)

Given oracle access to $f : [N] \to \{0, 1\}$, decide whether there exists an x such that $f(x) = 1$ with probability better than 2/3.

Theorem (Bennet, Bernstein, Brassard, Vazirani 1997)

For every quantum algorithm that makes $o(\sqrt{N})$ queries to f, there exists an f for which the algorithm fails to solve the Decision Grover Problem.

Thm. For every quantum algorithm that makes $o(\sqrt{N})$ queries to f, there exists an f for which the algorithm fails to solve the DGP.

Thm. For every quantum algorithm that makes $o(\sqrt{N})$ queries to f, there exists an f for which the algorithm fails to solve the DGP.

Proof Idea. Fix a T-query quantum algorithm:

$$\boldsymbol{Q}_f U_T \boldsymbol{Q}_f \cdots \boldsymbol{Q}_f U_3 \boldsymbol{Q}_f U_2 \boldsymbol{Q}_f U_1 |0^n\rangle$$

Thm. For every quantum algorithm that makes $o(\sqrt{N})$ queries to f, there exists an f for which the algorithm fails to solve the DGP.

Proof Idea. Fix a T-query quantum algorithm:

$$\boldsymbol{Q}_f U_T \boldsymbol{Q}_f \cdots \boldsymbol{Q}_f U_3 \boldsymbol{Q}_f U_2 \boldsymbol{Q}_f U_1 |0^n\rangle$$

If f is zero everywhere, $Q_f = I$.

Thm. For every quantum algorithm that makes $o(\sqrt{N})$ queries to f, there exists an f for which the algorithm fails to solve the DGP.

Proof Idea. Fix a T-query quantum algorithm:

$$\boldsymbol{Q}_f U_T \boldsymbol{Q}_f \cdots \boldsymbol{Q}_f U_3 \boldsymbol{Q}_f U_2 \boldsymbol{Q}_f U_1 |0^n\rangle$$

If f is zero everywhere, $Q_f = I$.

Interpolate between the non-zero case and the all-zero case...

Thm. For every quantum algorithm that makes $o(\sqrt{N})$ queries to f, there exists an f for which the algorithm fails to solve the DGP.

Proof Idea. Fix a T-query quantum algorithm:

$$\boldsymbol{Q}_f U_T \boldsymbol{Q}_f \cdots \boldsymbol{Q}_f U_3 \boldsymbol{Q}_f U_2 \boldsymbol{Q}_f U_1 |0^n\rangle$$

If f is zero everywhere, $Q_f = I$.

Interpolate between the non-zero case and the all-zero case...

$$|\phi^{x^*}\rangle = \boldsymbol{Q}_f U_T \boldsymbol{Q}_f \cdots \boldsymbol{Q}_f U_3 \boldsymbol{Q}_f U_2 \boldsymbol{Q}_f U_1 |0^n\rangle$$

Thm. For every quantum algorithm that makes $o(\sqrt{N})$ queries to f, there exists an f for which the algorithm fails to solve the DGP.

Proof Idea. Fix a T-query quantum algorithm:

$$\boldsymbol{Q}_f U_T \boldsymbol{Q}_f \cdots \boldsymbol{Q}_f U_3 \boldsymbol{Q}_f U_2 \boldsymbol{Q}_f U_1 |0^n\rangle$$

If f is zero everywhere, $Q_f = I$.

Interpolate between the non-zero case and the all-zero case...

$$|\phi^{x^*}\rangle = \boldsymbol{Q}_f U_T \boldsymbol{Q}_f \cdots \boldsymbol{Q}_f U_3 \boldsymbol{Q}_f U_2 \boldsymbol{Q}_f U_1 |0^n\rangle$$
$$\boldsymbol{Q}_f U_T \boldsymbol{Q}_f \cdots \boldsymbol{Q}_f U_3 \boldsymbol{Q}_f U_2 U_1 |0^n\rangle$$

Thm. For every quantum algorithm that makes $o(\sqrt{N})$ queries to f, there exists an f for which the algorithm fails to solve the DGP.

Proof Idea. Fix a T-query quantum algorithm:

$$\boldsymbol{Q}_f U_T \boldsymbol{Q}_f \cdots \boldsymbol{Q}_f U_3 \boldsymbol{Q}_f U_2 \boldsymbol{Q}_f U_1 |0^n\rangle$$

If f is zero everywhere, $Q_f = I$.

Interpolate between the non-zero case and the all-zero case...

$$\begin{aligned} |\phi^{x^*}\rangle = \; & \boldsymbol{Q}_f U_T \boldsymbol{Q}_f \cdots \boldsymbol{Q}_f U_3 \boldsymbol{Q}_f U_2 \boldsymbol{Q}_f U_1 |0^n\rangle \\ & \boldsymbol{Q}_f U_T \boldsymbol{Q}_f \cdots \boldsymbol{Q}_f U_3 \boldsymbol{Q}_f U_2 U_1 |0^n\rangle \\ & \boldsymbol{Q}_f U_T \boldsymbol{Q}_f \cdots \boldsymbol{Q}_f U_3 U_2 U_1 |0^n\rangle \end{aligned}$$

Thm. For every quantum algorithm that makes $o(\sqrt{N})$ queries to f, there exists an f for which the algorithm fails to solve the DGP.

Proof Idea. Fix a T-query quantum algorithm:

$$\boldsymbol{Q}_f U_T \boldsymbol{Q}_f \cdots \boldsymbol{Q}_f U_3 \boldsymbol{Q}_f U_2 \boldsymbol{Q}_f U_1 |0^n\rangle$$

If f is zero everywhere, $Q_f = I$.

Interpolate between the non-zero case and the all-zero case...

$$\begin{aligned}|\phi^{x^*}\rangle = \; & \boldsymbol{Q}_f U_T \boldsymbol{Q}_f \cdots \boldsymbol{Q}_f U_3 \boldsymbol{Q}_f U_2 \boldsymbol{Q}_f U_1 |0^n\rangle \\ & \boldsymbol{Q}_f U_T \boldsymbol{Q}_f \cdots \boldsymbol{Q}_f U_3 \boldsymbol{Q}_f U_2 U_1 |0^n\rangle \\ & \boldsymbol{Q}_f U_T \boldsymbol{Q}_f \cdots \boldsymbol{Q}_f U_3 U_2 U_1 |0^n\rangle \\ & \boldsymbol{Q}_f U_T \boldsymbol{Q}_f \cdots U_3 U_2 U_1 |0^n\rangle\end{aligned}$$

Thm. For every quantum algorithm that makes $o(\sqrt{N})$ queries to f, there exists an f for which the algorithm fails to solve the DGP.

Proof Idea. Fix a T-query quantum algorithm:

$$\boldsymbol{Q}_f U_T \boldsymbol{Q}_f \cdots \boldsymbol{Q}_f U_3 \boldsymbol{Q}_f U_2 \boldsymbol{Q}_f U_1 |0^n\rangle$$

If f is zero everywhere, $Q_f = I$.

Interpolate between the non-zero case and the all-zero case...

$$\begin{aligned}
|\phi^{x^*}\rangle = {} & \boldsymbol{Q}_f U_T \boldsymbol{Q}_f \cdots \boldsymbol{Q}_f U_3 \boldsymbol{Q}_f U_2 \boldsymbol{Q}_f U_1 |0^n\rangle \\
& \boldsymbol{Q}_f U_T \boldsymbol{Q}_f \cdots \boldsymbol{Q}_f U_3 \boldsymbol{Q}_f U_2 U_1 |0^n\rangle \\
& \boldsymbol{Q}_f U_T \boldsymbol{Q}_f \cdots \boldsymbol{Q}_f U_3 U_2 U_1 |0^n\rangle \\
& \boldsymbol{Q}_f U_T \boldsymbol{Q}_f \cdots U_3 U_2 U_1 |0^n\rangle \\
& \vdots
\end{aligned}$$

Thm. For every quantum algorithm that makes $o(\sqrt{N})$ queries to f, there exists an f for which the algorithm fails to solve the DGP.

Proof Idea. Fix a T-query quantum algorithm:

$$\boldsymbol{Q}_f U_T \boldsymbol{Q}_f \cdots \boldsymbol{Q}_f U_3 \boldsymbol{Q}_f U_2 \boldsymbol{Q}_f U_1 |0^n\rangle$$

If f is zero everywhere, $Q_f = I$.

Interpolate between the non-zero case and the all-zero case. . .

$$\begin{aligned}
|\phi^{x^*}\rangle = \; & \boldsymbol{Q}_f U_T \boldsymbol{Q}_f \cdots \boldsymbol{Q}_f U_3 \boldsymbol{Q}_f U_2 \boldsymbol{Q}_f U_1 |0^n\rangle \\
& \boldsymbol{Q}_f U_T \boldsymbol{Q}_f \cdots \boldsymbol{Q}_f U_3 \boldsymbol{Q}_f U_2 U_1 |0^n\rangle \\
& \boldsymbol{Q}_f U_T \boldsymbol{Q}_f \cdots \boldsymbol{Q}_f U_3 U_2 U_1 |0^n\rangle \\
& \boldsymbol{Q}_f U_T \boldsymbol{Q}_f \cdots U_3 U_2 U_1 |0^n\rangle \\
& \vdots \\
|\phi\rangle = \; & U_T \cdots U_3 U_2 U_1 |0^n\rangle
\end{aligned}$$

Proof Idea (cont'd).

$$\sum_x \alpha_{x,t}|x\rangle = \text{state before } t\text{-th query}$$

$$x^* = \text{the ``target'' value}$$

Proof Idea (cont'd).

$$\sum_x \alpha_{x,t}|x\rangle = \text{state before } t\text{-th query}$$

$$x^* = \text{the "target" value}$$

- With each query, the Euclidean distance between the two states can grow by at most $2|\alpha_{x^*,t}|$.

Proof Idea (cont'd).

$$\sum_x \alpha_{x,t}|x\rangle = \text{state before } t\text{-th query}$$

$$x^* = \text{the "target" value}$$

- With each query, the Euclidean distance between the two states can grow by at most $2|\alpha_{x^*,t}|$.
- To distinguish, the distance after T queries needs to be at least a constant ϵ, so: $\epsilon \leq 2\sum_{t=1}^{T}|\alpha_{x^*,t}|$.

Proof Idea (cont'd).

$$\sum_x \alpha_{x,t}|x\rangle = \text{state before } t\text{-th query}$$

$$x^* = \text{the "target" value}$$

- With each query, the Euclidean distance between the two states can grow by at most $2|\alpha_{x^*,t}|$.
- To distinguish, the distance after T queries needs to be at least a constant ϵ, so: $\epsilon \leq 2\sum_{t=1}^{T}|\alpha_{x^*,t}|$.
- To complete the proof, sum over all N possible x^*s:

$$\epsilon N \leq 2\sum_{t=1}^{T}\sum_{x^*=1}^{N}|\alpha_{x^*,t}| \leq \sum_{t=1}^{T}\sqrt{N}\sqrt{\sum_{x^*=1}^{N}|\alpha_{x^*,t}|^2} \leq 2T\sqrt{N}.$$

Proof Idea (cont'd).

$$\sum_x \alpha_{x,t}|x\rangle = \text{state before } t\text{-th query}$$

$$x^* = \text{the ``target'' value}$$

- With each query, the Euclidean distance between the two states can grow by at most $2|\alpha_{x^*,t}|$.
- To distinguish, the distance after T queries needs to be at least a constant ϵ, so: $\epsilon \leq 2\sum_{t=1}^{T}|\alpha_{x^*,t}|$.
- To complete the proof, sum over all N possible x^*s:

$$\epsilon N \leq 2\sum_{t=1}^{T}\sum_{x^*=1}^{N}|\alpha_{x^*,t}| \leq \sum_{t=1}^{T}\sqrt{N}\sqrt{\sum_{x^*=1}^{N}|\alpha_{x^*,t}|^2} \leq 2T\sqrt{N}.$$

$$\Rightarrow \quad \frac{\epsilon}{2}\sqrt{N} \leq T$$

Overview

Motivation

Background

Grover's Algorithm

Applications
- Breaking Block Ciphers
- Collision Finding
- Password Cracking

Conclusion

Breaking Block Ciphers

For this talk, a block cipher is an efficient deterministic function:

$$E : \mathcal{K} \times \{0,1\}^n \rightarrow \{0,1\}^n.$$

Breaking Block Ciphers

For this talk, a block cipher is an efficient deterministic function:

$$E : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n.$$

A necessary (not sufficient) security property is that, for $k \stackrel{R}{\leftarrow} \mathcal{K}$, an adversary given

$$E(k, \text{“0”}), E(k, \text{“1”}), E(k, \text{“2”})$$

cannot recover k faster than a brute-force search of the key-space.

Breaking Block Ciphers

For this talk, a block cipher is an efficient deterministic function:

$$E : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n.$$

A necessary (not sufficient) security property is that, for $k \xleftarrow{R} \mathcal{K}$, an adversary given

$$E(k, \text{“0”}), E(k, \text{“1”}), E(k, \text{“2”})$$

cannot recover k faster than a brute-force search of the key-space.

Viewing $E(\cdot,\cdot)$ as an oracle, an adversary making q queries should succeed with probability at most $\approx q/|\mathcal{K}|$.

Breaking Block Ciphers

Grover search recovers the key in time $O(\sqrt{|\mathcal{K}|})$.

Breaking Block Ciphers

Grover search recovers the key in time $O(\sqrt{|\mathcal{K}|})$.

Attack Using Grover

1. Attacker receives challenge $c = (c_0, c_1, c_2)$.
2. Define a function $f_c : \mathcal{K} \to \{0,1\}$ as:

$$f_c(k) \stackrel{\text{def}}{=} \{(E(k, \text{“0”}), E(k, \text{“1”}), E(k, \text{“2”})) = (c_0, c_1, c_2)\}.$$

3. Run Grover's algorithm on f_c.
4. In $O(\sqrt{|\mathcal{K}|})$ iterations, Grover returns k w.h.p.

Breaking Block Ciphers

Grover search recovers the key in time $O(\sqrt{|\mathcal{K}|})$.

Attack Using Grover

1. Attacker receives challenge $c = (c_0, c_1, c_2)$.
2. Define a function $f_c : \mathcal{K} \to \{0, 1\}$ as:

$$f_c(k) \stackrel{\text{def}}{=} \{(E(k, \text{“0”}), E(k, \text{“1”}), E(k, \text{“2”})) = (c_0, c_1, c_2)\}.$$

3. Run Grover's algorithm on f_c.
4. In $O(\sqrt{|\mathcal{K}|})$ iterations, Grover returns k w.h.p.

Attacking AES-128

Breaking Block Ciphers

Grover search recovers the key in time $O(\sqrt{|\mathcal{K}|})$.

Attack Using Grover

1. Attacker receives challenge $c = (c_0, c_1, c_2)$.
2. Define a function $f_c : \mathcal{K} \to \{0,1\}$ as:

$$f_c(k) \stackrel{\text{def}}{=} \{(E(k, \text{“0”}), E(k, \text{“1”}), E(k, \text{“2”})) = (c_0, c_1, c_2)\}.$$

3. Run Grover's algorithm on f_c.
4. In $O(\sqrt{|\mathcal{K}|})$ iterations, Grover returns k w.h.p.

Attacking AES-128

Special-purpose classical attack: $2^{126.1}$ (Bogdanov et al. 2011)

Breaking Block Ciphers

Grover search recovers the key in time $O(\sqrt{|\mathcal{K}|})$.

Attack Using Grover

1. Attacker receives challenge $c = (c_0, c_1, c_2)$.
2. Define a function $f_c : \mathcal{K} \to \{0,1\}$ as:

$$f_c(k) \stackrel{\text{def}}{=} \{(E(k, \text{“0”}), E(k, \text{“1”}), E(k, \text{“2”})) = (c_0, c_1, c_2)\}.$$

3. Run Grover's algorithm on f_c.
4. In $O(\sqrt{|\mathcal{K}|})$ iterations, Grover returns k w.h.p.

Attacking AES-128

Special-purpose classical attack: $2^{126.1}$ (Bogdanov et al. 2011)
Generic quantum attack: 2^{64}. **!!!**

Hash Collisions

Let H be a random function.

Hash Collisions

Let H be a random function.

Problem: Given oracle access to $H : [2N] \rightarrow [N]$, find distinct elements x and x' such that $H(x) = H(x')$.

Hash Collisions

Let H be a random function.

Problem: Given oracle access to $H : [2N] \to [N]$, find distinct elements x and x' such that $H(x) = H(x')$.

To succeed with constant probability (by the Birthday Bound), a classical algorithm requires $\Theta(\sqrt{N})$ queries.

[Compute $H(0), H(1), H(2), \ldots$ until you find a collision.]

Hash Collisions

Let H be a random function.

Problem: Given oracle access to $H : [2N] \to [N]$, find distinct elements x and x' such that $H(x) = H(x')$.

To succeed with constant probability (by the Birthday Bound), a classical algorithm requires $\Theta(\sqrt{N})$ queries.

[Compute $H(0), H(1), H(2), \ldots$ until you find a collision.]

Theorem (Brassard, Høyer, Tapp 1997)

There is a quantum collision-finding algorithm that makes $O(N^{1/3})$ quantum queries and succeeds with constant probability.

Quantum Collision Finding

Algorithm Idea

Quantum Collision Finding

Algorithm Idea

- Build a big table of random values and their hashes.

Quantum Collision Finding

Algorithm Idea

- Build a big table of random values and their hashes.

r_0	$H(r_0)$
r_1	$H(r_1)$
r_2	$H(r_2)$
r_3	$H(r_3)$
$\vdots$	$\vdots$

$O(N^{1/3})$

Quantum Collision Finding

Algorithm Idea

- Build a big table of random values and their hashes.
- Use Grover search to quickly find a value that collides with one in the table.

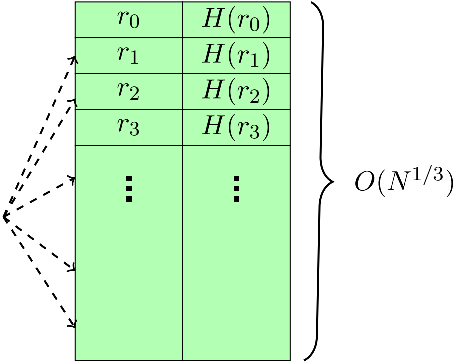
r_0	$H(r_0)$
r_1	$H(r_1)$
r_2	$H(r_2)$
r_3	$H(r_3)$
$\vdots$	$\vdots$

$O(N^{1/3})$

Quantum Collision Finding

Algorithm Idea

- Build a big table of random values and their hashes.
- Use Grover search to quickly find a value that collides with one in the table.



Quantum Collision Finding

Algorithm

1. Sample $O(N^{1/3})$ random integers $r_i \in [2N]$, compute $h_i \leftarrow H(r_i)$, and store each (r_i, h_i) in a table T.
2. Define a function $f_T : [2N] \to \{0,1\}$:

$$f_T(x) \stackrel{\text{def}}{=} \begin{cases} h^* \leftarrow H(x) \\ \text{Look for a pair } (r_i, h_i) \in T \text{ with } h_i = h^* \\ \text{If such a pair exists and } r_i \neq x \text{, return } 1. \end{cases}$$

3. Use Grover search to find a “good” x.
4. Use the table to find the colliding r, and output (x, r).

Quantum Collision Finding

Algorithm

1. Sample $O(N^{1/3})$ random integers $r_i \in [2N]$, compute $h_i \leftarrow H(r_i)$, and store each (r_i, h_i) in a table T.
2. Define a function $f_T : [2N] \to \{0,1\}$:

$$f_T(x) \stackrel{\text{def}}{=} \begin{cases} h^* \leftarrow H(x) \\ \text{Look for a pair } (r_i, h_i) \in T \text{ with } h_i = h^* \\ \text{If such a pair exists and } r_i \neq x \text{, return } 1. \end{cases}$$

3. Use Grover search to find a "good" x.
4. Use the table to find the colliding r, and output (x, r).

Analysis

- Step 1 makes $O(N^{1/3})$ queries to H.

Quantum Collision Finding

Algorithm

1. Sample $O(N^{1/3})$ random integers $r_i \in [2N]$, compute $h_i \leftarrow H(r_i)$, and store each (r_i, h_i) in a table T.
2. Define a function $f_T : [2N] \to \{0,1\}$:

$$f_T(x) \stackrel{\text{def}}{=} \begin{cases} h^* \leftarrow H(x) \\ \text{Look for a pair } (r_i, h_i) \in T \text{ with } h_i = h^* \\ \text{If such a pair exists and } r_i \neq x\text{, return } 1. \end{cases}$$

3. Use Grover search to find a "good" x.
4. Use the table to find the colliding r, and output (x, r).

Analysis

- Step 1 makes $O(N^{1/3})$ queries to H.
- Step 3 is a Grover search over space of size $2N$, with $\approx N^{1/3}$ possible solutions.

Quantum Collision Finding

Algorithm

1. Sample $O(N^{1/3})$ random integers $r_i \in [2N]$, compute $h_i \leftarrow H(r_i)$, and store each (r_i, h_i) in a table T.
2. Define a function $f_T : [2N] \to \{0,1\}$:

$$f_T(x) \stackrel{\text{def}}{=} \begin{cases} h^* \leftarrow H(x) \\ \text{Look for a pair } (r_i, h_i) \in T \text{ with } h_i = h^* \\ \text{If such a pair exists and } r_i \neq x\text{, return } 1. \end{cases}$$

3. Use Grover search to find a “good” x.
4. Use the table to find the colliding r, and output (x, r).

Analysis

- Step 1 makes $O(N^{1/3})$ queries to H.
- Step 3 is a Grover search over space of size $2N$, with $\approx N^{1/3}$ possible solutions. $\Rightarrow O(\sqrt{N/N^{1/3}}) = O(N^{1/3})$ queries.

Collision Finding in Practice

Is the collision-finding algorithm practical?

Collision Finding in Practice

Is the collision-finding algorithm practical?

- The **query** complexity is $O(N^{1/3})$.

Collision Finding in Practice

Is the collision-finding algorithm practical?

- The **query** complexity is $O(N^{1/3})$. ✓

Collision Finding in Practice

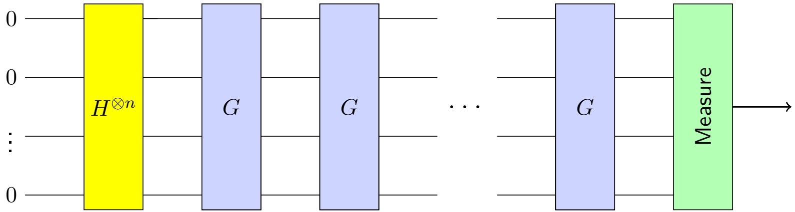
Is the collision-finding algorithm practical?

- The **query** complexity is $O(N^{1/3})$. ✓
- What is the size of the quantum circuit?

Collision Finding in Practice

Is the collision-finding algorithm practical?

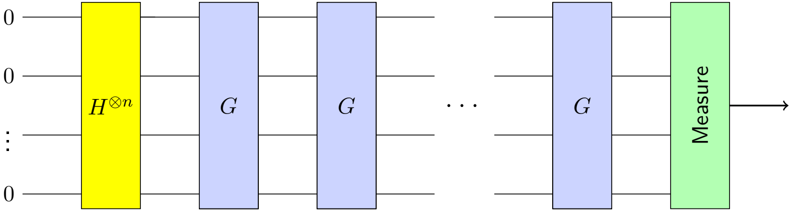
- The **query** complexity is $O(N^{1/3})$. ✓
- What is the size of the quantum circuit?



Collision Finding in Practice

Is the collision-finding algorithm practical?

- The **query** complexity is $O(N^{1/3})$. ✓
- What is the size of the quantum circuit?



Each Grover iteration encodes a table of size $\Theta(N^{1/3})$, so the G circuit has $\Theta(N^{1/3})$ gates. (!)

Collision Finding in Practice

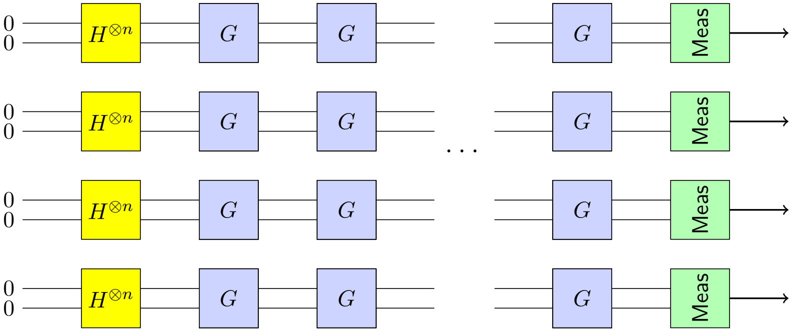
- Mounting the attack requires a QC with $\Theta(N^{1/3})$ qubits!
  (In contrast, the cipher attack requires a QC with a few thousand qubits.)

Collision Finding in Practice

- Mounting the attack requires a QC with $\Theta(N^{1/3})$ qubits!
  (In contrast, the cipher attack requires a QC with a few thousand qubits.)
- If you have $\Theta(N^{1/3})$ qubits, you might as well use **parallel Grover** search:

Collision Finding in Practice

- Mounting the attack requires a QC with $\Theta(N^{1/3})$ qubits!
  (In contrast, the cipher attack requires a QC with a few thousand qubits.)
- If you have $\Theta(N^{1/3})$ qubits, you might as well use **parallel Grover** search:



Collision Finding in Practice

Parallel Grover (Grover and Rudolph 2003)

1. Pick an $x_0 \xleftarrow{R} [N]$.
2. Define $f : [2N] \to \{0, 1\}$ as:

$$f_{x_0}(x) \stackrel{\text{def}}{=} \{H(x) = H(x_0) \text{ and } x \neq x_0\}.$$

3. Divide search space into $N^{1/3}$ pieces.
4. Run Grover on each piece in parallel.

Collision Finding in Practice

Parallel Grover (Grover and Rudolph 2003)

1. Pick an $x_0 \xleftarrow{R} [N]$.
2. Define $f : [2N] \to \{0, 1\}$ as:

$$f_{x_0}(x) \stackrel{\text{def}}{=} \{H(x) = H(x_0) \text{ and } x \neq x_0\}.$$

3. Divide search space into $N^{1/3}$ pieces.
4. Run Grover on each piece in parallel.

Analysis.

Each machine searches over a space of size $O(N/N^{1/3})$.

Collision Finding in Practice

Parallel Grover (Grover and Rudolph 2003)

1. Pick an $x_0 \xleftarrow{R} [N]$.
2. Define $f : [2N] \to \{0, 1\}$ as:

$$f_{x_0}(x) \stackrel{\text{def}}{=} \{H(x) = H(x_0) \text{ and } x \neq x_0\}.$$

3. Divide search space into $N^{1/3}$ pieces.
4. Run Grover on each piece in parallel.

Analysis.

Each machine searches over a space of size $O(N/N^{1/3})$.
We expect one space to contain a colliding input.

Collision Finding in Practice

Parallel Grover (Grover and Rudolph 2003)

1. Pick an $x_0 \xleftarrow{R} [N]$.
2. Define $f : [2N] \rightarrow \{0, 1\}$ as:

$$f_{x_0}(x) \stackrel{\text{def}}{=} \{H(x) = H(x_0) \text{ and } x \neq x_0\}.$$

3. Divide search space into $N^{1/3}$ pieces.
4. Run Grover on each piece in parallel.

Analysis.

Each machine searches over a space of size $O(N/N^{1/3})$.
We expect one space to contain a colliding input.
Running time is $O(\sqrt{N^{2/3}}) = O(N^{1/3})$.

Collision Finding in Practice

Parallel Grover (Grover and Rudolph 2003)

1. Pick an $x_0 \xleftarrow{R} [N]$.
2. Define $f : [2N] \to \{0,1\}$ as:

$$f_{x_0}(x) \stackrel{\text{def}}{=} \{H(x) = H(x_0) \text{ and } x \neq x_0\}.$$

3. Divide search space into $N^{1/3}$ pieces.
4. Run Grover on each piece in parallel.

Analysis.

Each machine searches over a space of size $O(N/N^{1/3})$.
We expect one space to contain a colliding input.
Running time is $O(\sqrt{N^{2/3}}) = O(N^{1/3})$.

If you have a size-$\Theta(N^{1/3})$ **classical** computer, finding collisions with the parallel rho method only takes time $O(N^{1/6})$!
(Van Oorschot and Wiener 1999) (Bernstein 2009)

Password Cracking

Modern OSes store passwords as $H(\texttt{salt}, \texttt{password})$, where:

- H is a "moderately hard" function, and
- `salt` is a random string.

Password Cracking

Modern OSes store passwords as $H(\texttt{salt}, \texttt{password})$, where:

- H is a "moderately hard" function, and
- `salt` is a random string.

User	**Password**
alice	cardinal650
bob	Stanford!
carol	CSRulez
	⋮

Password Cracking

Modern OSes store passwords as $H(\texttt{salt}, \texttt{password})$, where:

- H is a "moderately hard" function, and
- `salt` is a random string.

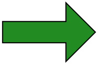
User	Password
alice	cardinal650
bob	Stanford!
carol	CSRulez
	⋮

Password Cracking

Modern OSes store passwords as $H(\texttt{salt}, \texttt{password})$, where:

- H is a "moderately hard" function, and
- `salt` is a random string.

User	**Password**
alice	cardinal650
bob	Stanford!
carol	CSRulez
	⋮



User	**Salt**	**HashedPass**
alice	0x0738	0x89d7f1a
bob	0xaab3	0x1704193
carol	0x9c3e	0x726ebd9
	⋮	

Password Cracking

Modern OSes store passwords as $H(\texttt{salt}, \texttt{password})$, where:

- H is a "moderately hard" function, and
- `salt` is a random string.

User	Password
alice	cardinal650
bob	Stanford!
carol	CSRulez
	⋮

User	Salt	HashedPass
alice	0x0738	0x89d7f1a
bob	0xaab3	0x1704193
carol	0x9c3e	0x726ebd9
	⋮	

If someone steals your password file, they have to do some work ("password cracking") to recover the stored passwords.

Password Cracking

Problem: Given oracle access to $H : [N] \to [N]$, a dictionary of candidate passwords

$$\mathcal{D} = \{\texttt{password, 12345, qwerty, ...}\} \subseteq [N],$$

and a target τ, find an $x \in \mathcal{D}$ such that $H(x) = \tau$.

Password Cracking

Problem: Given oracle access to $H : [N] \to [N]$, a dictionary of candidate passwords

$$\mathcal{D} = \{\texttt{password},\ \texttt{12345},\ \texttt{qwerty},\ \ldots\} \subseteq [N],$$

and a target τ, find an $x \in \mathcal{D}$ such that $H(x) = \tau$.

Inverting a function with *hints*.

Password Cracking

Problem: Given oracle access to $H : [N] \to [N]$, a dictionary of candidate passwords

$$\mathcal{D} = \{\texttt{password, 12345, qwerty, ...}\} \subseteq [N],$$

and a target τ, find an $x \in \mathcal{D}$ such that $H(x) = \tau$.

Password Cracking

Problem: Given oracle access to $H : [N] \to [N]$, a dictionary of candidate passwords

$$\mathcal{D} = \{\texttt{password, 12345, qwerty, ...}\} \subseteq [N],$$

and a target τ, find an $x \in \mathcal{D}$ such that $H(x) = \tau$.

Classical attack: $\Theta(|\mathcal{D}|)$ queries to H (to succeed w.p. $1/2$)

Password Cracking

Problem: Given oracle access to $H : [N] \to [N]$, a dictionary of candidate passwords

$$\mathcal{D} = \{\texttt{password},\ \texttt{12345},\ \texttt{qwerty},\ \ldots\} \subseteq [N],$$

and a target τ, find an $x \in \mathcal{D}$ such that $H(x) = \tau$.

Classical attack: $\Theta(|\mathcal{D}|)$ queries to H (to succeed w.p. $1/2$)

Grover search: $O(\sqrt{|\mathcal{D}|})$ attack.* **(New?)**

Password Cracking

Problem: Given oracle access to $H : [N] \to [N]$, a dictionary of candidate passwords

$$\mathcal{D} = \{\texttt{password, 12345, qwerty, ...}\} \subseteq [N],$$

and a target τ, find an $x \in \mathcal{D}$ such that $H(x) = \tau$.

Classical attack: $\Theta(|\mathcal{D}|)$ queries to H (to succeed w.p. $1/2$)

Grover search: $O(\sqrt{|\mathcal{D}|})$ attack.* **(New?)**

Quantum computers essentially break all password hashing functions.

Quantum Password Cracking

1. **Define** a function $f_{\mathcal{D}} : \{1, 2, \ldots, |\mathcal{D}|\} \to \{0, 1\}$ as:

$$f_{\mathcal{D}}(i) \stackrel{\text{def}}{=} \begin{cases} d_i \leftarrow \text{“}i\text{th entry in dictionary } \mathcal{D}\text{”} \\ \text{return } \tau \stackrel{?}{=} H(d_i) \end{cases}$$

2. **Run** Grover search to find a “good” i.

Search will use $O(\sqrt{|\mathcal{D}|})$ queries to H and $\mathcal{D}$.

Quantum Password Cracking

1. **Define** a function $f_\mathcal{D} : \{1, 2, \ldots, |\mathcal{D}|\} \to \{0, 1\}$ as:

$$f_\mathcal{D}(i) \stackrel{\text{def}}{=} \begin{cases} d_i \leftarrow \text{“}i\text{th entry in dictionary } \mathcal{D}\text{”} \\ \text{return } \tau \stackrel{?}{=} H(d_i) \end{cases}$$

2. **Run** Grover search to find a “good” i.

Search will use $O(\sqrt{|\mathcal{D}|})$ queries to H and $\mathcal{D}$.

- $\mathcal{C}_H =$ Cost of H query.
- $\mathcal{C}_\mathcal{D} =$ Cost of $\mathcal{D}$ query.

Quantum Password Cracking

1. **Define** a function $f_{\mathcal{D}} : \{1, 2, \ldots, |\mathcal{D}|\} \to \{0, 1\}$ as:

$$f_{\mathcal{D}}(i) \stackrel{\text{def}}{=} \begin{cases} d_i \leftarrow \text{"}i\text{th entry in dictionary } \mathcal{D}\text{"} \\ \text{return } \tau \stackrel{?}{=} H(d_i) \end{cases}$$

2. **Run** Grover search to find a "good" i.

Search will use $O(\sqrt{|\mathcal{D}|})$ queries to H and $\mathcal{D}$.

- $\mathcal{C}_H =$ Cost of H query.
- $\mathcal{C}_{\mathcal{D}} =$ Cost of $\mathcal{D}$ query.

$$\text{Attack cost} = (\#\text{ iterations}) \cdot (\text{Cost per iteration})$$

Quantum Password Cracking

1. **Define** a function $f_{\mathcal{D}} : \{1, 2, \ldots, |\mathcal{D}|\} \to \{0, 1\}$ as:

$$f_{\mathcal{D}}(i) \stackrel{\text{def}}{=} \begin{cases} d_i \leftarrow \text{"}i\text{th entry in dictionary } \mathcal{D}\text{"} \\ \text{return } \tau \stackrel{?}{=} H(d_i) \end{cases}$$

2. **Run** Grover search to find a "good" i.

Search will use $O(\sqrt{|\mathcal{D}|})$ queries to H and $\mathcal{D}$.

- $\mathcal{C}_H =$ Cost of H query.
- $\mathcal{C}_{\mathcal{D}} =$ Cost of $\mathcal{D}$ query.

$$\begin{aligned} \text{Attack cost} &= (\#\text{ iterations}) \cdot (\text{Cost per iteration}) \\ &\approx \sqrt{|\mathcal{D}|}(\mathcal{C}_H + \mathcal{C}_{\mathcal{D}}) \end{aligned}$$

Quantum Password Cracking

1. **Define** a function $f_{\mathcal{D}} : \{1, 2, \ldots, |\mathcal{D}|\} \to \{0, 1\}$ as:

$$f_{\mathcal{D}}(i) \stackrel{\text{def}}{=} \begin{cases} d_i \leftarrow \text{“}i\text{th entry in dictionary } \mathcal{D}\text{”} \\ \text{return } \tau \stackrel{?}{=} H(d_i) \end{cases}$$

2. **Run** Grover search to find a “good” i.

Search will use $O(\sqrt{|\mathcal{D}|})$ queries to H and $\mathcal{D}$.

- $\mathcal{C}_H =$ Cost of H query.
- $\mathcal{C}_{\mathcal{D}} =$ Cost of $\mathcal{D}$ query.

$$\begin{aligned} \text{Attack cost} &= (\#\text{ iterations}) \cdot (\text{Cost per iteration}) \\ &\approx \sqrt{|\mathcal{D}|}(\mathcal{C}_H + \mathcal{C}_{\mathcal{D}}) \quad (\text{Could be} \approx |\mathcal{D}| \log N) \end{aligned}$$

Quantum Password Cracking

1. **Define** a function $f_{\mathcal{D}} : \{1, 2, \ldots, |\mathcal{D}|\} \to \{0, 1\}$ as:

$$f_{\mathcal{D}}(i) \stackrel{\text{def}}{=} \begin{cases} d_i \leftarrow \text{"}i\text{th entry in dictionary } \mathcal{D}\text{"} \\ \text{return } \tau \stackrel{?}{=} H(d_i) \end{cases}$$

2. **Run** Grover search to find a "good" i.

Search will use $O(\sqrt{|\mathcal{D}|})$ queries to H and $\mathcal{D}$.

- $\mathcal{C}_H =$ Cost of H query.
- $\mathcal{C}_{\mathcal{D}} =$ Cost of $\mathcal{D}$ query.

$$\begin{aligned} \text{Attack cost} &= (\#\text{ iterations}) \cdot (\text{Cost per iteration}) \\ &\approx \sqrt{|\mathcal{D}|}(\mathcal{C}_H + \mathcal{C}_{\mathcal{D}}) \quad (\text{Could be } \approx |\mathcal{D}| \log N) \\ &\approx |\mathcal{D}|^{3/2} + \sqrt{\mathcal{D}} \cdot \mathcal{C}_H \end{aligned}$$

Quantum Password Cracking

1. **Define** a function $f_{\mathcal{D}} : \{1, 2, \ldots, |\mathcal{D}|\} \to \{0, 1\}$ as:

$$f_{\mathcal{D}}(i) \stackrel{\text{def}}{=} \begin{cases} d_i \leftarrow \text{“}i\text{th entry in dictionary } \mathcal{D}\text{”} \\ \text{return } \tau \stackrel{?}{=} H(d_i) \end{cases}$$

2. **Run** Grover search to find a “good” i.

Search will use $O(\sqrt{|\mathcal{D}|})$ queries to H and $\mathcal{D}$.

- $\mathcal{C}_H =$ Cost of H query.
- $\mathcal{C}_{\mathcal{D}} =$ Cost of $\mathcal{D}$ query.

$$\begin{aligned} \text{Attack cost} &= (\#\text{ iterations}) \cdot (\text{Cost per iteration}) \\ &\approx \sqrt{|\mathcal{D}|}(\mathcal{C}_H + \mathcal{C}_{\mathcal{D}}) \quad (\text{Could be} \approx |\mathcal{D}| \log N) \\ &\approx |\mathcal{D}|^{3/2} + \sqrt{\mathcal{D}} \cdot \mathcal{C}_H \end{aligned}$$

This often beats the classical $|\mathcal{D}| \cdot \mathcal{C}_H$ attack!

Quantum Password Cracking

If we can represent the dictionary $\mathcal{D}$ with a **small circuit**, then the quantum attack is devastating:

$$|\mathcal{D}| \cdot \mathcal{C}_H \qquad \text{decreases to} \qquad \approx \sqrt{|\mathcal{D}|} \cdot \mathcal{C}_H.$$

Quantum Password Cracking

If we can represent the dictionary $\mathcal{D}$ with a **small circuit**, then the quantum attack is devastating:

$$|\mathcal{D}| \cdot \mathcal{C}_H \qquad \text{decreases to} \qquad \approx \sqrt{|\mathcal{D}|} \cdot \mathcal{C}_H.$$

Using **amplitude amplification** (Brassard, Høyer, Mosca, Tapp 2002), we can generalize the attack from

password dictionaries to *password distributions*.

The End of Password Hashing?

Say that an attacker's budget allows for 2^{24} hash computations. . .

The End of Password Hashing?

Say that an attacker's budget allows for 2^{24} hash computations...

Type	Len	Classical	Quantum
Lower-case alpha	6 char	2^{28}	2^{14}
	8 char	2^{37}	2^{19}
	10 char	2^{47}	2^{24}
Alphanumeric	6 char	2^{36}	2^{18}
	8 char	2^{47}	2^{23}
	10 char	2^{60}	2^{30}
Printable ASCII	6 char	2^{39}	2^{20}
	8 char	2^{52}	2^{26}
	10 char	2^{66}	2^{33}

Overview

Motivation

Background

Grover's Algorithm

Applications

Conclusion

Conclusions

Quantum computers can solve black-box search problems faster than classical computers can.

Conclusions

Quantum computers can solve black-box search problems faster than classical computers can.

Future Directions

Conclusions

Quantum computers can solve black-box search problems faster than classical computers can.

Future Directions

1. **Find** quantum collision-finding algorithms that beat the classical ones, in terms of qubit complexity. (Grover and Rudolph 2003)

Conclusions

Quantum computers can solve black-box search problems faster than classical computers can.

Future Directions

1. **Find** quantum collision-finding algorithms that beat the classical ones, in terms of qubit complexity. (Grover and Rudolph 2003)
   - ... or prove that none exist.

Conclusions

Quantum computers can solve black-box search problems faster than classical computers can.

Future Directions

1. **Find** quantum collision-finding algorithms that beat the classical ones, in terms of qubit complexity. (Grover and Rudolph 2003)
   - . . . or prove that none exist.
2. **Cryptanalyze** proposed post-quantum cryptosystems. Switching from RSA $\rightarrow$ LWE doesn't necessarily protect you.

Conclusions

Quantum computers can solve black-box search problems faster than classical computers can.

Future Directions

1. **Find** quantum collision-finding algorithms that beat the classical ones, in terms of qubit complexity. (Grover and Rudolph 2003)
    - . . . or prove that none exist.
2. **Cryptanalyze** proposed post-quantum cryptosystems. Switching from RSA $\rightarrow$ LWE doesn't necessarily protect you.
3. **Prove** time-space lower bounds for quantum algorithms in the random-oracle model.

Conclusions

Quantum computers can solve black-box search problems faster than classical computers can.

Future Directions

1. **Find** quantum collision-finding algorithms that beat the classical ones, in terms of qubit complexity. (Grover and Rudolph 2003)
   - ...or prove that none exist.
2. **Cryptanalyze** proposed post-quantum cryptosystems. Switching from RSA $\rightarrow$ LWE doesn't necessarily protect you.
3. **Prove** time-space lower bounds for quantum algorithms in the random-oracle model.

Thank you!

References

Background

- Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach.*
- Michael Nielsen and Isaac Chuang. *Quantum Computation and Quantum Information.*
- John Watrous. Lecture notes: *Introduction to Quantum Computing* `https://cs.uwaterloo.ca/~watrous/LectureNotes.html`

Grover's Algorithm

- Lov Grover. "A Fast Quantum Mechanical Algorithm for Database Search" (1996).
  `https://arxiv.org/abs/quant-ph/9605043`
- Michel Boyer, Gilles Brassard, Peter Høyer, and Alain Tapp. "Tight Bounds on Quantum Searching" (1996).
  `https://arxiv.org/abs/quant-ph/9605034`
- Richard Jozsa. "Searching in Grover's Algorithm" (1999).
  `https://arxiv.org/pdf/quant-ph/9901021`
  Gives the nice geometric interpretation of Grover search.
- Gilles Brassard, Peter Høyer, Michele Mosca, and Alain Tapp. "Quantum Amplitude Amplification and Estimation" (2000).
  `https://arxiv.org/abs/quant-ph/0005055`

References

Lower Bound

- Charles H. Bennett, Ethan Bernstein, Gilles Brassard, and Umesh Vazirani "Strengths and Weaknesses of Quantum Computing" (1997). `https://arxiv.org/abs/quant-ph/9701001`
- Ronald de Wolf. Lecture notes: "Quantum Lower Bounds" (2005). `http://www.iro.umontreal.ca/~tappa/Summer%20School/montreal05.pdf`
- Scott Aaronson. Lecture notes: "6.845: Quantum Complexity Theory" (2009). `https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-845-quantum-complexity-theory-fall-2010/lecture-notes/`

Collision Finding

- Gilles Brassard, Peter Høyer, and Alain Tapp. "Quantum Algorithm for the Collision Problem" (1997). `https://arxiv.org/abs/quant-ph/9705002`
- Paul van Oorschot and Michael J. Wiener. "Parallel Collision Search with Cryptanalytic Applications" (1999). `http://people.scs.carleton.ca/~paulv/papers/JoC97.pdf`
- Lov Grover and Terry Rudolph. "How Significant are the Known Collision and Element Distinctness Quantum Algorithms?" (2003). `http://arxiv.org/pdf/quant-ph/0306017`
- Daniel J. Bernstein. "Cost Analysis of Hash Collisions: Will Quantum Computers Make SHARCs Obsolete?" (2009). `http://cr.yp.to/hash/collisioncost-20090823.pdf`

References

AES Cryptanalysis

- Andrey Bogdanov, Dmitry Khovratovich, and Christian Rechberger. "Biclique Cryptanalysis of the Full AES" (2011). `http://research.microsoft.com/en-us/projects/cryptanalysis/aesbc.pdf`

Analysis of Grover's Algorithm

Claim

The operator $R = H^{\otimes n} Q_0 H^{\otimes n}$ reflects over the hyperplane orthogonal to $|h\rangle$.

Analysis of Grover's Algorithm

Claim

The operator $R = H^{\otimes n} Q_0 H^{\otimes n}$ reflects over the hyperplane orthogonal to $|h\rangle$.

The Q_0 operator flips the sign of $|0^n\rangle$ in a superposition:

$$Q_0 = I - 2|0^n\rangle\langle 0^n|.$$

Analysis of Grover's Algorithm

Claim

The operator $R = H^{\otimes n} Q_0 H^{\otimes n}$ reflects over the hyperplane orthogonal to $|h\rangle$.

The Q_0 operator flips the sign of $|0^n\rangle$ in a superposition:

$$Q_0 = I - 2\underbrace{|0^n\rangle\langle 0^n|}_{\text{outer product}}.$$

Analysis of Grover's Algorithm

Claim

The operator $R = H^{\otimes n} Q_0 H^{\otimes n}$ reflects over the hyperplane orthogonal to $|h\rangle$.

The Q_0 operator flips the sign of $|0^n\rangle$ in a superposition:

$$Q_0 = I - 2|0^n\rangle\langle 0^n|.$$

Analysis of Grover's Algorithm

Claim

The operator $R = H^{\otimes n} Q_0 H^{\otimes n}$ reflects over the hyperplane orthogonal to $|h\rangle$.

The Q_0 operator flips the sign of $|0^n\rangle$ in a superposition:

$$Q_0 = I - 2|0^n\rangle\langle 0^n|.$$

Then $R = H^{\otimes n} Q_0 H^{\otimes n} = I - 2|h\rangle\langle h|$, so R takes:

$$|h\rangle \mapsto -|h\rangle \qquad \text{and} \qquad |h^\perp\rangle \mapsto |h^\perp\rangle.$$

Analysis of Grover's Algorithm

Claim

The operator $R = H^{\otimes n} Q_0 H^{\otimes n}$ reflects over the hyperplane orthogonal to $|h\rangle$.

The Q_0 operator flips the sign of $|0^n\rangle$ in a superposition:

$$Q_0 = I - 2|0^n\rangle\langle 0^n|.$$

Then $R = H^{\otimes n} Q_0 H^{\otimes n} = I - 2|h\rangle\langle h|$, so R takes:

$$|h\rangle \mapsto -|h\rangle \qquad \text{and} \qquad |h^\perp\rangle \mapsto |h^\perp\rangle.$$

So, for any vector $|v\rangle = \alpha|h\rangle + \beta|h^\perp\rangle$, R maps:

$$\alpha|h\rangle + \beta|h^\perp\rangle \qquad \mapsto \qquad -\alpha|h\rangle + \beta|h^\perp\rangle.$$

$\square$