A quantum polynomial time search algorithm for certain unsorted finite lists

Stéphane Lemieux
Department of Computer Science
MacEwan University, Edmonton, AB, T5J 4S2
lemieuxs5@macewan.ca

July 22, 2022

Abstract

Grover famously showed that any unsorted list, of finite size N, can be searched in $\mathrm{O}(\sqrt{N})$ time via quantum computation. Bennett et. al. demonstrated that any algorithm general enough to search any finite unsorted list must take at least $\mathrm{O}(\sqrt{N})$ time via quantum computation. We demonstrate a quantum algorithm that can search a proper subclass of finite, unsorted lists, of size N, in a time that is polynomial in $log(N)$. We demonstrate how it can be used to search the keyspace of any block cipher that can be implemented on a quantum computer with the keyspace in superpositon. In particular we give a polynomial time attack on $AES-128$, $AES-192$ and $AES-256$.

1 Introduction

Any unstructured, i.e. unsorted, finite list of length N can be searched on a classical computer in $O(N)$ time. In [3], Grover famously showed that any finite unstructured list can be sorted via quantum computation in $O(\sqrt{N})$ time. As well, Bennett, et. al. [1], showed that no algorithm that can search any unsorted list of size N, can do so in $o(\sqrt{N})$ via quantum computation. Taken together, the latter two results appear to completely answer the question: How fast can one search a finite unstructured list via quantum computation? However, neither result precludes the existence of a faster search algorithm that is only successful on a proper, but useful, subclass of lists.

In this work, we demonstrate a quantum algorithm that can successfully search certain unsorted lists of size N in polynomial time in $log\ N$. The

algorithm requires $O(log\ N)$ calls to an oracle similar to the one used by Grover, followed by $O(log\ N)$ multiplications by the exact diffusion matrix that Grover uses, and finally $O(log\ N)$ Hadamard (element-wise) products of two separable N-state quantum systems.

We show that the subclass of searchable lists includes all permutations of B_n, where B_n denotes the set of all binary strings of a length n. We give a generalization that can be used to search the keyspace of any block cipher, provided it can be implemented on a quantum computer with the entire keyspace in superpositon. In particular we give a polynomial time attack that successfully recovers the secret key of $AES-128$, $AES-192$ and $AES-256$.

1.1 Format of the Paper

In **Section 2**, we introduce notation and revisit Grover's algorithm as motivation for the rest of the paper.

In **Section 3**, we introduce a new search algorithm that takes as input, any list of size $N = 2^N$ whose elements form a permutation of B_n, the set of all binary strings of length n where n is an even positive integer.

In **Section 4**, we give a small example to demonstrate the algorithm of Section 3.

In **Section 5**, we detail the adjustments to the algorithm of Section 3, that allow it to accommodate a wider variety of lists.

In **Section 6**, we demonstrate that any block cipher, that can be performed via quantum computing with the set of all secret keys in superposition, can be searched in polynomial time in the length of the key.

2 Notation and Summary of Grover's Algorithm

We will represent a superposition of quantum states as

$$\sum_i a_i \left|S_i\right\rangle$$

Where the amplitudes a_i are complex numbers such that $\Sigma_i \|a_i\|^2 = 1$ and each $\left|S_i\right\rangle$ is a basis state.

Algorithm 3.1 below, draws significantly from Grover's, so we revisit it first. Recall that Grover assumes we have an unstructured list L of size N with a unique element $w \in L$ whose position in L we wish to determine. L will correspond to a superposition of quantum states, $\Sigma_i a_i \left|S_i\right\rangle$ such that w is represented by some unique state S_v.

Grover initializes the quantum system to

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

He then executes the loop $O(\sqrt{N})$ times:

1. Apply an oracle that recognizes w and rotates the phase of the quantum state by π radians so that its amplitude a_v is replaced by $-a_v$ and the amplitudes of all other quantum states are left unaltered.

2. Multiply the resulting system, as a column vector, by the diffusion matrix D given by $D_{ii} = -1 + \frac{2}{N}$ and $D_{ij} = \frac{2}{N}$ for all $i \neq j$.

Each application of the loop increases the amplitude a_v while simultaneously decreasing the amplitude of all other states. After $O(\sqrt{N})$ applications of the loop, we are guaranteed that $a_v \geq \frac{1}{\sqrt{2}}$. When the final step, measuring the system, is executed the probability that S_v is returned is $a_v^2 \geq \frac{1}{2}$.

3 A Quantum Polynomial Time Search Algorithm

It helps to visualize Algorithm 3.1, at least initially, as applying only to lists L that are permutations of B_n, where n is an even positive integer. Thus L will have size $N = 2^n$, with n even. We will broaden the application to other lists later though we will never reach the full generality of Grover's Algorithm.

Algorithm 3.1

Step 1: Initialize $\frac{n}{2}$ Separable Systems The algorithm starts by initializing $\frac{n}{2}$ separable quantum systems, each with superposition

$$Q_j = \sum_{i=0}^{N-1} \frac{1}{\sqrt{N}} |S_{ij}\rangle , \text{for } j \in \{0, 1, ... \frac{n}{2} - 1\}$$

To be clear, fixing j gives us a quantum system with N states. For each system Q_j, the state S_{vj} will correspond to the position of w but there is no entanglement between systems. I.e. Q_i and Q_j are separable for each $i \neq j$. This set up is analogous to distinct people running a search for the same element w of the same list L, each using a separate quantum computer.

Step 2: Phase Shifts Algorithm,3.1 only requires one application of an oracle to each system. Recall that Grover's algorithm has an oracle that recognizes w and shifts the phase of its corresponding quantum state. Let $(w_0 w_1 \cdots w_{n-1})_2$ be the binary representation of w. For each system Q_j, we use a unique oracle C_j that recognizes the substring $(w_{2j} w_{2j+1})_2$. We perform a phase shift of π radians replacing a_{ij} with $-a_{ij}$ if the binary string $x \in L$, corresponding to a_{ij} satisfies $(w_{2j} w_{2j+1})_2 = (x_{2j} x_{2j+1})_2$. Since L is a permutation of all binary strings of length n, the oracle C_j will shift the phase of exactly $\frac{N}{4}$ states. After the phase shifts, Q_j will have

- $\frac{N}{4}$ states of the form $\frac{-1}{\sqrt{N}} |S_{ij}\rangle$ and
- $\frac{3N}{4}$ states of the form $\frac{1}{\sqrt{N}} |S_{ij}\rangle$.

Step 3: Multiply by Grover's Diffusion Matrix We multiply each quantum system by Grover's Diffusion Matrix, D, only once. The $3:1$ ratio, of states without a phase shift to states with a phase shift, is a critical property that allows us to limit the application of Grover's loop to once per system. We claim that after taking the product $Q'_j = D \times Q_j$, the resulting system has the following form:

- Each state, of Q_j, of the form $\frac{1}{\sqrt{N}} |S_{ij}\rangle$ will be replaced, in Q'_j, by the state $0 |S_{ij}\rangle$
- Each state, of Q_j, of the form $\frac{-1}{\sqrt{N}} |S_{ij}\rangle$ will be replaced, in Q'_j, by $\frac{2}{\sqrt{N}} |S_{ij}\rangle$

Proof: For each system Q_j we define p_j to be the proportion of states, whose amplitude undergoes a phase shift. Then the number of states, whose amplitude undergoes a phase shift is $p_j N$, while $(1 - p_j)N$ do not. Then, after taking the products $D \times Q_j$, the states whose amplitude did not undergo a phase shift will have new amplitude:

$$\frac{1}{\sqrt{N}}(-1 + \frac{2}{N}) + \frac{1}{\sqrt{N}}((1 - p_j)N - 1)(\frac{2}{N}) + \frac{-1}{\sqrt{N}}(p_j N)(\frac{2}{N}) = \frac{1 - 4p_j}{\sqrt{N}}$$

Similarly, the states of Q_j that did undergo a phase shift will have new amplitude

$$\frac{-1}{\sqrt{N}}(-1 + \frac{2}{N}) + \frac{1}{\sqrt{N}}((1 - p_j)N)(\frac{2}{N}) + \frac{-1}{\sqrt{N}}(p_j N - 1)(\frac{2}{N}) = \frac{3 - 4p_j}{\sqrt{N}}$$

Each two bit substring has four possible values: $(00)_2$, $(01)_2$, $(10)_2$ and $(11)_2$, only one of which will result in a phase shift for a given Q_j. Since

each of these substrings must appear equally often, in a permutation of B_n, we must have $p_j = 0.25$ and the result follows immediately. $\boxdot$

We will consider the case where $p_j \neq 0.25$ later, but remark now that performance degrades. This is the key compromise that Grover's Algorithm does not require.

Step 4: Take Hadamard (element-wise) Products The next step is to take successive Hadammard products, denoted $\odot$ as follows. See [5] for details on Hadamard products.

- $H_1 = Q'_0 \odot Q'_1$ and
- $H_i = H_{i-1} \odot Q'_i$, for $i \in \{2, 3, \ldots, \frac{n}{2} - 1\}$

We demonstrate that in the final system $H = H_{\frac{n}{2}-1}$, the state corresponding to the position of w in L will be the only state with non-zero amplitude.

Proof

For each $j \in \{0, 1, \ldots, \frac{n}{2} - 1\}$, the amplitude of the quantum state in each Q_j that corresponds to w, will undergo the phase shift because it will be recongnized by each oracle C_j. Consequently, the corresponding state in each Q'_j will be non-zero, $(\frac{2}{\sqrt{N}})$, and thus the amplitude of the Hadamard product of these states will be non-zero. In contrast, for each $x \in L$, $x \neq w$ implies that in at least one Q_j, the amplitude of the state that corresponds to x will not undergo a phase shift. Consequently this amplitude will be 0 in Q'_j. When the amplitudes of the states that correspond to x, in each Q_j are multipied together, the resulting product must also be 0. $\boxdot$

Step 5: Measure the System The final step then is to measure the quantum system H which has only a single non-zero amplitude which corresponds to the index of w in L as desired.

4 Small Example

For extra clarity, we walk through a small example where $N = 2^4 = 16$.

Suppose L is the following permutation of B_4 and also suppose that we are looking for the position of $w = (0111)_2$. For the remainder of the example, we will highlight the position of w in L and in each quantum system to make the explanation easier to follow.

$$L = \{(0110)_2, (0010)_2, (1100)_2, (1110)_2, (0000)_2, (0011)_2, (1101)_2, (1001)_2,$$

$$(1111)_2, (0100)_2, (1011)_2, (1010)_2, (\mathbf{0111})_\mathbf{2}, (0101)_2, (0001)_2, (1000)_2\}$$

Note that we do not want the algorithm to return $(0111)_2$ but rather its position in L. The quantum states are indexed from 0, which corresponds to $|0000\rangle$, to 15, which corresponds to $|1111\rangle$. The element w is then in position 12 which corresponds to $|1100\rangle$. Thus we expect the algorithm to measure $|1100\rangle$ and return $(1100)_2$ in the final step.

Step 1: Initialize Q_j

We initialize Q_0 and Q_1 as

$$\begin{aligned} Q_0 = (&\frac{1}{4}|0000\rangle + \frac{1}{4}|0001\rangle + \frac{1}{4}|0010\rangle + \frac{1}{4}|0011\rangle + \\ &\frac{1}{4}|0100\rangle + \frac{1}{4}|0101\rangle + \frac{1}{4}|0110\rangle + \frac{1}{4}|0111\rangle + \\ &\frac{1}{4}|1000\rangle + \frac{1}{4}|1001\rangle + \frac{1}{4}|1010\rangle + \frac{1}{4}|1011\rangle + \\ &\frac{1}{4}|\mathbf{1100}\rangle + \frac{1}{4}|1101\rangle + \frac{1}{4}|1110\rangle + \frac{1}{4}|1111\rangle) \end{aligned}$$

$$\begin{aligned} Q_1 = (&\frac{1}{4}|0000\rangle + \frac{1}{4}|0001\rangle + \frac{1}{4}|0010\rangle + \frac{1}{4}|0011\rangle + \\ &\frac{1}{4}|0100\rangle + \frac{1}{4}|0101\rangle + \frac{1}{4}|0110\rangle + \frac{1}{4}|0111\rangle + \\ &\frac{1}{4}|1000\rangle + \frac{1}{4}|1001\rangle + \frac{1}{4}|1010\rangle + \frac{1}{4}|1011\rangle + \\ &\frac{1}{4}|\mathbf{1100}\rangle + \frac{1}{4}|1101\rangle + \frac{1}{4}|1110\rangle + \frac{1}{4}|1111\rangle) \end{aligned}$$

Step 2: Apply Phase Shifts

We chose $w = (0111)_2$ so the oracle C_0 will run through L and shift the phase of the amplitude of any state in Q_0 that corresponds to an element in L that begins with $(01)_2$ the first two bits of w. These states are positions 0, 9, 12, and 13. Likewise, the oracle C_1 will run through L and shift the phase of the amplitude of any state in Q_1 that corresponds to an element in L that ends with $(11)_2$, the last two bits of w. These states are the positions 5, 8, 10, and 12. After applying the oracles we have:

$$\begin{aligned} Q_0 = (&\frac{-1}{4}|0000\rangle + \frac{1}{4}|0001\rangle + \frac{1}{4}|0010\rangle + \frac{1}{4}|0011\rangle + \\ &\frac{1}{4}|0100\rangle + \frac{1}{4}|0101\rangle + \frac{1}{4}|0110\rangle + \frac{1}{4}|0111\rangle + \\ &\frac{1}{4}|1000\rangle + \frac{-1}{4}|1001\rangle + \frac{1}{4}|1010\rangle + \frac{1}{4}|1011\rangle + \end{aligned}$$

$$\frac{-1}{4}|\mathbf{1100}\rangle + \frac{-1}{4}|1101\rangle + \frac{1}{4}|1110\rangle + \frac{1}{4}|1111\rangle)$$

$$Q_1 = (\frac{1}{4}|0000\rangle + \frac{1}{4}|0001\rangle + \frac{1}{4}|0010\rangle + \frac{1}{4}|0011\rangle +$$
$$\frac{1}{4}|0100\rangle + \frac{-1}{4}|0101\rangle + \frac{1}{4}|0110\rangle + \frac{1}{4}|0111\rangle +$$
$$\frac{-1}{4}|1000\rangle + \frac{1}{4}|1001\rangle + \frac{-1}{4}|1010\rangle + \frac{1}{4}|1011\rangle +$$
$$\frac{-1}{4}|\mathbf{1100}\rangle + \frac{1}{4}|1101\rangle + \frac{1}{4}|1110\rangle + \frac{1}{4}|1111\rangle)$$

Step 3: Multiply by Diffusion Matrix D
The diffusion matrix is:

$$D = \begin{bmatrix} \text{-}7/8 & 1/8 & 1/8 & \dots & 1/8 \\ 1/8 & \text{-}7/8 & 1/8 & \dots & 1/8 \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 1/8 & 1/8 & 1/8 & \dots & \text{-}7/8 \end{bmatrix}$$

And the resulting $Q'_0 = D \times Q_0$ is

$$Q'_0 = (\frac{1}{2}|0000\rangle + 0|0001\rangle + 0|0010\rangle + 0|0011\rangle +$$
$$0|0100\rangle + 0|0101\rangle + 0|0110\rangle + 0|0111\rangle +$$
$$0|1000\rangle + \frac{1}{2}|1001\rangle + 0|1010\rangle + 0|1011\rangle +$$
$$\frac{1}{2}|\mathbf{1100}\rangle + \frac{1}{2}|1101\rangle + 0|1110\rangle + 0|1111\rangle)$$

Similarly the resulting $Q'_1 = D \times Q_1$ is

$$Q'_1 = (0|0000\rangle + 0|0001\rangle + 0|0010\rangle + 0|0011\rangle +$$
$$0|0100\rangle + \frac{1}{2}|0101\rangle + 0|0110\rangle + 0|0111\rangle +$$
$$\frac{1}{2}|1000\rangle + 0|1001\rangle + \frac{1}{2}|1010\rangle + 0|1011\rangle +$$
$$\frac{1}{2}|\mathbf{1100}\rangle + 0|1101\rangle + 0|1110\rangle + 0|1111\rangle)$$

Step 4: Take Hadamard Products For this small example $H_1 = Q_1' \times Q_2'$ is the only Hadamard product necessary. After normalizing, we get:

$$\begin{aligned} H_1 = (&0\,|0000\rangle + 0\,|0001\rangle + 0\,|0010\rangle + 0\,|0011\rangle + \\ &0\,|0100\rangle + 0\,|0101\rangle + 0\,|0110\rangle + 0\,|0111\rangle + \\ &0\,|1000\rangle + 0\,|1001\rangle + 0\,|1010\rangle + 0\,|1011\rangle + \\ &1\,|\mathbf{1100}\rangle + 0\,|1101\rangle + 0\,|1110\rangle + 0\,|1111\rangle) \end{aligned}$$

Step 5: Measure the Resulting System
Measuring H_1 will return the index 1100, as desired.

5 Searchable Lists

As noted in the introduction, Algorithm 3.1 runs faster than Grover's but cannot be used on all finite lists. In the previous sections we demonstrated that it can be used on lists which are permutations of the B_n, for any even positive integer n. In this section we look at other lists on which our Algorithm 3.1 can be applied, with probability of success at least 0.5. Modifications to the list L, may be necessary first. In particular we may need to embed L in a large list with more favourable properties. We allow this as long as we can still retrieve the index of w in the original list.

Definition 1 (Searchable List) *Suppose L is a list and w is an element of L. If Algorithm 3.1 is effective at searching for the position of w in L with probability of success at least 0.5 then we will call L a searchable list with respect to w or simply a searchable list if the context is clear.*

As the definition suggests, some lists may be searchable for with respect to some elements but not others. We will investigate this later, but suffice to say that the suitability depends on the probabilities p_j which are specific to the choice of w.

5.1 Case 1: Searchable Lists

A list of L of length N, consisting of binary strings, is searchable with respect to element w if the following sufficient but not necessary properties are satisfied.

1. **Even Power Property:** $N = 2^n$ for some even positive integer n

2. **Proportion Property:** Recall from step 3 of Algorithm 3.1, for w we define p_j to be the portion of the elements of L whose $2j^{th}$ and $2j+1^{th}$ bits match $(w_{2j}w_{2j+1})_2$. We require that $p_j = 0.25$ for each $j \in \{0, 1, \ldots, \frac{n}{2} - 1\}$.

The above properties are sufficient for Algorithm 3.1 to be effective. As mentioned, we make no claim that these properties are necessary, only that they are sufficient. It also follows immediately that for any permutation of B_n and for any string $w \in B_n$ the above properties hold as long as n is even. Finally we note that L need not contain all of B_n. We can substitute random binary strings of length n, as long as the proportion property remains satisfied.

We now explore the cases where list L, with element w, can be embedded in a searchable list in such a way that our Algorithm 3.1 can be used to efficiently deduce the index of w in L with probability at least 0.5.

5.2 Case 2: n is Odd:

Suppose L is a list of length $N = 2^n$, where n is an odd integer and w is an element of L for which the proportion property is satisfied in the following respect:

For each of the following 2-bit substrings

$$\{(w_1w_2)_2, (w_3w_4)_2, \ldots, (w_{n-2}w_{n-1})_2\},$$

we have $p_j = 0.25$ and $(w_0)_2 = (x_0)_2$ for exactly $\frac{1}{2}$ of the elements $x \in L$. Here, p_j is re-indexed so that p_0 refers to $(w_1w_2)_2$, p_1 refers to $(w_3w_4)_2$, etc.

We proposed two general ways to deal with this setup.

1. Option 1 is to ignore the most significant bit of each string, and run Algorithm 3.1 with the oracles restricted to

   $$\{(w_1w_2)_2, (w_3w_4)_2, \ldots, (w_{n-2}w_{n-1})_2\}.$$

   If we do, there will be exactly two states of H with non-zero amplitude, either of which can be returned upon measuring in Step 5, one of which corresponds to w. However, the indicies of the two states with non-zero amplitude will differ by exaclty 2^{n-1}, so we only have to run the algorithm once.

2. Option 2 is to embed L in a list L' of size 2^{n+1}. We do so by making two copies of L. We append each x in the first copy of L, to 0 and in

the second copy to 1. We then run Algoirthm 3.1 and search for $0w$. This is equivalent to taking the tensor product of a single extra qubit and the n-qubit system we normally initialize.

5.3 Case 3: $2^{n-1} < N < 2^n$:

Given the possible transformation from Case 2, we can assume n is even. We also assume that L satisfies the proportion property with respect to w.

In this case we build a new list L' by including a sufficient number of uniformly random binary strings to ensure that L' has size 2^n. The chance that we add one or more copies of w to L' is less than 0.5 since L is larger than 2^{n-1}. Let A be the event that w is unique in L', and A' denote its complement. Let B be the event that measuring the final system returns the correct index for w. Then

$$p(B) = p(B \mid A)p(A) + p(B \mid A')p(A').$$

$p(B \mid A)p(A) > 1(0.5) = 0.5$ and $p(B \mid A')p(A') > 0$ so the Algorithm 3.1 will return the correct index with probability > 0.5. Furthermore, if Algorithm 3.1 returns the wrong index, it will be clearly larger than N, so we will know to repeat the experiment.

It is worth noting that if we add uniformly generated binary strings we are not guaranteed that the proportion property will be preserved though the deviation from 0.25 of any p_j should be minute. We deal with this case in the next subsection.

5.4 Case 4: Proportion Property Fails

We begin by noting that if the deviations of the p_j from 0.25 are small enough then we may not need to make any correction at all. The first lemma below shows that if such a deviation is small enough, relative to the size of N, then the amplitude of the state corresponding to w will be less than 1 but still greater than $\frac{1}{\sqrt{2}}$ so the probability of the algorithm being successful will still be at least 0.5.

The second point we make is that for the cryptoanalytic applications we discuss in later sections, we will not need to worry that the proportion property may not be satisfied because if this were the case, the encryption algorithm would likely fall to a statistical attack. For this reason, great care is taken in the design of cryptographic algorithms to ensure that such statistical attacks are not effective.

To be balanced in our exposition, the third point we make is that if any proportion p_j deviates form 0.25 by too much, the list may be unusable for the chosen w. For example, if $p_j > 0.5$ then the amplitudes that undergo phase shift in Step 2 of Algorithm 3.1 will be smaller in magnitude, after multiplication by D, than those that did not undergo a phase shift.

However, if the deviation from 0.25 is limited to a small number of p_j and it is predictable which p_j will deviate significantly from 0.25, then it is possible to simply omit the corresponding Q'_j's from Step 4, of Algorithm 3.1, where we take the Hadamard products. For each $Q'j$ we omit, the probability of selecting the correct w from L will decrease by a factor of 4. For example in the small example above, if we omit Q_0, then measuring the final system will return any one of the 4 non-zero amplitudes of Q'_1, each with equal probability. As long as the number of $Q'j$ that need to be left out, does not increase as N increases, we will have a lower bound on the probability of selecting the correct final state and we can simply repeat Algorithm 3.1 enough times to ensure the probability of success is greater than 0.5.

The fourth and final point we make is that the embedding lemma described below, may itself be time consuming: so much so that the embedding process, combined with algorithm 3.1 may offer no speed improvement over Grover's Algorithm.

We now return to the case where one or more of the proportions p_j, deviates from 0.25 by some value $-0.25 \ll \epsilon_j \ll 0.25$. Then the states in Q'_j, that did not undergo a phase shift in Step 2, will have small but non-zero amplitudes. Then, even if w is unique, the final state corresponding to w in H will not have amplitude 1. However, we need to show that if the ϵ_j's are small enough then the amplitude of the state in H that corresponds to w is at least $\frac{1}{\sqrt{2}}$. First we start by comparing the magnitude of the amplitude of the state that corresponds to w with the next largest magnitude of amplitude in the final system.

Lemma 1 *The magnitude of the amplitude of the state corresponding to w in H and the next largest amplitude will be at least $\frac{2-\epsilon_j}{\epsilon_j}$.*

Proof

The states in Q'_j that did not undergo a phase shift will have amplitude

$$\frac{1 - 4(0.25 \pm \epsilon_j)}{\sqrt{N}} = \frac{\pm\epsilon_j}{\sqrt{N}}$$

and states that did undergo a phase shift will have amplitude

$$\frac{1-4(0.25\pm\epsilon_j)}{\sqrt{N}} = \frac{2\pm\epsilon_j}{\sqrt{N}}.$$

For any $x \in L$ such that $x \neq w$, there exists at least one $j \in \{0, 1, \ldots, \frac{n}{2}-1\}$ such that the substring $x_{2j}x_{2j+1}$ will not induce a phase shift from oracle C_j. Since w will always induce phase shifts, the amplitude in the final state of H corresponding to x will be smaller than the amplitude of the final state corresponding to w buy a factor of at least:

$$\frac{\frac{2\pm\epsilon_j}{\sqrt{N}}}{\frac{\pm\epsilon_j}{\sqrt{N}}} = \frac{2\pm\epsilon_j}{\pm\epsilon_j} \geq \frac{2-\epsilon_j}{\epsilon_j} \ \boxdot$$

Lemma 2 *Suppose the maximum amplitude, in H, other than the the amplitude corresponding to w is a. Then the amplitude corresponding to w is at least $a\frac{2-\epsilon_j}{\epsilon_j}$. If $\epsilon_j < \frac{2}{\sqrt{N}+1}$, then the amplitude corresponding to w is at least $\frac{1}{\sqrt{2}}$.*

Proof The amplitude of each state in H that does not correspond to the position of w will be at most a. Thus the sum of the square amplitudes that do not correspond to w is at most $(N-1)a^2$. The square of the amplitude that does correspond to w is

$$(a\frac{2-\epsilon_j}{\epsilon_j})^2 > (a\frac{2-\frac{2}{\sqrt{N}+1}}{\frac{2}{\sqrt{N}+1}})^2 = a^2(N-1)^2.$$

Since these two values must sum to 1 the result holds. $\boxdot$

Of course assuming that all other amplitudes in H will be a is does not lead to the best possible restriction of e_j, but the previous result does show that if we can get each p_j close enough to 0.25, then Algorithm 3.1 will work with probability at least 0.5. The next result shows that we can always embed a list L, for which $0 < p_j < 0.5$, in a larger list L' such that if $p_j \neq 0.25$, then the resulting proportion p'_j is as close to 0.25 as we require.

Lemma 3 *Suppose L is a list of size $N = 2^n$, where n is even. Suppose also that we are searching for w in L and at least one $p_j \neq 0.25$. We can embed L in a list L' of size $N' = 2^{n+2k}$ such that the corresponding proportion will be $p'_j = \frac{2k-1}{2k}(0.25) + \frac{1}{2k}(p_j)$.*

Proof We concatenate the $2k$-bit string $(00\ldots00)_2$ to each binary string $x \in L$. For every $2k$-bit string not equal to $(00\ldots00)_2$, we concatenate the string with a uniformly random n-bit string. We repeat this process N times until there are a total of $N' = 2^{n+2k}$, including the original list.

For example, if $k = 1$, we concatenate $(00)_2$ to each $x \in L$ and then concatenate each of $(01)_2$, $(10)_2$, and $(11)_2$ to a separate list of N uniformly generate binary strings of length N. So we end up with $4N$ binary strings of length $n + 2$.

In the above example, we are searching for $(00w_0w_1\ldots w_{n-1})_2$ which is unique in L' if w is unique L. Furthermore, for L', $p'_0 = 0.25$, and for each $j \in \{1, 2, \ldots \frac{n}{2}\}$, $p'_j = \frac{3}{4}(0.25) + \frac{1}{4}p_j$ which is closer to 0.25 than p_j was. $\boxdot$

6 Cryptanalytic Applications: Block Ciphers

The problem we address in this section is as follows: Suppose we have a block cipher scheme, public or private key,

$$E : B_{n'} \times B_n \to B_n, \text{ via } E(k, m) = c,$$

where k is the secret key, m is the plaintext message, and c is the encrypted ciphertext. Suppose we also have a small number of plaintext, ciphertext pairs (m_i, c_i) which were all encrypted via the same secret key k. We would like to determine the secret key k.

The key space need not be the same size as the message space but we will begin with the case where $n' = n$.

For any fixed key, k, the mapping $E(k, \cdot) : M \to C$ is necessarily a bijective map between the message space and the ciphertext space. However, if instead we fix the plaintext message m then the mapping $E(\cdot, m) : K \to C$ need not be injective. However, can assume that there is no bias in favour of key-collisions of the form $E(k_1, m) = E(k_2, m)$ occuring more or less likely than we would expect by random chance, for the same reason we can assume the proportion property will be satified. This makes it possible to determine the probability, for a fixed message m, whether or not a chosen key k_1 will have any other key k_2 that collides with it on encryption of m. We will discuss this further after detailing the following search algorithm to recover k.

Shor [4] demonstrates that, to solve the discrete logarithm problem, we can form the following quantum system in superposition:

$$\frac{1}{P-1}\sum_{a=0}^{P-2}\sum_{b=0}^{P-2}|a,b,g^a x^{-b}(mod\ P)\rangle$$

We can think of the above system as being the encryption of x as $g^a x^{-b}(mod\ P)$, where a, b is the secret key. From this perspective, the set system is set up with every possible secret key in superposition, and the corresponding encryption of the same message x appended. We include this as an illustrative example for Algorithm 7.1.

As perhaps the quintessential example, we note that [2] already tested the resilience of AES to exhaustive search via Grover's algorithm. We demonstrate a similar procedure to testing AES with Algorithm 7.1 below.

Algorithm 7.1

Step 1 Initialize $\frac{n}{2}$ systems, $\forall j \in \{0, 1, \ldots, \frac{n}{2} - 1\}$, as follows:

$$Q_j = \frac{1}{\sqrt{2^n}}\sum_{k=0}^{2^n-1}|k, E(k, m_0)\rangle$$

Step 2 Use oracles C_j based on ciphertext c_0 on each appropriate 2-qubit pair in $E(k, m_0)$ to determine which amplitudes should incur a phase shift of π radians.

Step 3 For each system Q_j, multiply the amplitudes by Grover's diffusion matrix D.

Step 4 For each system Q'_j that results from Step 3, take successive Hadamard products as before.

Step 5 Measure the Resulting System.

We now show that the probability that the above Agorithm 7.1 returns the appropriate key is at least $\frac{1}{2}$ by separately considering the cases where there are no key collisions, exactly one key collision and more than one key collisions.

Case1: No Key Collisions for $(\mathbf{k_1}, \mathbf{m_0})$

Consider the key k_1 that we are searching for as fixed. If we encrypt $E(k_1, m_0) = c_0$, we wish to know the probability that there are is other key k_2 such that $E(k_1, m_0) = E(k_2, m_0)$.

We proceed by noting that for each key k_i, we can consider $E(k_i, \cdot)$ to be a permutation of B_n and hence an element of the symmetric group on $N = 2^n$ elements, which we will label $Sym(N)$. Note that $|Sym(N)| = N!$

but only N of those elements correspond to encryption keys. The message m_0, has N possible images, via $Sym(N)$. Thus, by the orbit stabalizer theorem, we must have that exactly $\frac{1}{N}$ of the elements of $Sym(N)$ that map m_0 to c_0. So the probability that a particular key k_2 does not encrypt m_0 to c_0 is $1 - \frac{1}{N}$. By extension, the probability than no other key k_i, $i \neq 1$ encrypts m_0 to c_0 is:

$$(1 - \frac{1}{N})^{N-1} \approx \frac{1}{e} \times \frac{1}{1 - \frac{1}{N}} \approx \frac{1}{e}$$

The approximations are appropriate but N is generally of the size 2^{128} to 2^{256} is current cryptographic protocols.

Case 2: Exactly One Key Collision

We also want to consider the probability that there is exactly one other key k_2 such that $E(k_1, m_0) = E(k_2, m_0)$. If this case occurs, then the algorithm will still return the correct key with probabitity 0.5. The probability, of exactly one collision is the binomial distribution term with $N - 1$ choice, probability $\frac{1}{N}$ that each key collides, and probability $1 - \frac{1}{N}$ that each key does not collide. This gives

$$\binom{N-1}{1}(\frac{1}{N})(1 - \frac{1}{N})^{N-2} \approx (1 - \frac{1}{N})^{N-1} \approx \frac{1}{e}.$$

Case 3: More than One Key Collisions By the law of total probability, the probability that there are more than one collisions is $1 - \frac{2}{e}$.

Even if there are more than one key collisions, the probability that algorithm 7.1 correctly returns k_1, is some $q > 0$. Thus the probability that the search algorithm returns the correct key k is at least:

$$\frac{1}{e}(1) + \frac{1}{e}(\frac{1}{2}) + (1 - \frac{2}{e})q > \frac{3}{2}(\frac{1}{e}) > \frac{1}{2}.$$

It will be left to future work to apply Algorithm 7.1 to specific encryption algorithms.

It remains to deal with cases where the the keyspace differs in size from the plaintext space. If the keyspace is smaller than the message space it should not pose any problem to Algorithm 7.1 and the chances of a key-collision are greatly reduced.

If the keyspace is larger, as for example AES has message sapce 2^{128} and can run with keyspaces of size 2^{128}, 2^{192}, and 2^{256}, we simply encrypt more

than one message block with each key and concatenate them. For example, for either 192, and 256-bit AES, we initialize the systems as follows:

$$Q_j = \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} |k, E(k, m_0)E(k, m_1)\rangle \, .$$

The rest of Algorithm 7.1 will be as before. The probability of key collisions for 256 -bit AES will be the same as for 128-bit and will be almost 0 for 128-bit AES.

7 Conclusion

We have shown that the search Algorithm 3.1 runs faster than Grover's Algorithm but only accepts as input a smaller subclass of unsorted finite lists. However, the subclass of searchable lists is broad enough to include the keyspace, plaintext space and ciphertext space of block ciphers. It remains for future work to demonstrate which encryption algorithms, public and private key, are susceptible to the quantum cryptanaltyic attack we have presented, but note that AES falls to such an attack.

References

[1] Bennett, C.H.; Bernstein, E.; Brassard, G.; Vazirani, U. "Strengths and weaknesses of quantum computing", *Siam Journal on computing,* 26, No. 5. https://doi.org/10.1137/S009753979630093

[2] Bonnetain, X.; Naya-Plasencia, M.; Schrottenloher, A., "Quantum Security Analysis of AES", *IACR Transactions on Symmetric Cryptology,* 2019(2), 55–93. https://doi.org/10.13154/tosc.v2019.i2.55-93

[3] Grover, L.K. "A fast quantum mechanical algorithm for database search", *Proceedings of the twenty-eighth annual ACM symposium on theory of computing* 1996, 212-219

[4] Shor, Peter W. "Polynomial-Time algorithms for prime factorization and discrete logarithms on a quantum computer", *SIAM Review* 41 No. 2 (1999), 303-332

[5] Zhao, L., Zhao, Z., Rebentrost, P., Fitzsimons, J. "Compiling basic linear algebra subroutines for quantum computers", *Quantum Machine intelligence* (2021) 3:21, https: doi.org/10.1007/s42484-021-00048-8