CS 258: Quantum Cryptography (Fall 2025)
Homework 2 (100 points)

1 Problem 1 (25 points)

Wiesner's quantum money scheme used the set of states $\{|0\rangle, |1\rangle, |+\rangle, |-\rangle\}$. We said in class that such states can be copied with probability 3/4, in the sense that verifying both copies will accept with that probability. In order to get the success probability of a would-be cloner down to be exponentially-small, we actually have banknotes be many independent states from the set. Effectively, doing so is utilizing higher-dimensional states over the joint system in order to decrease the adversary's success probability.

A natural question is whether we can get the adversary's success probability down by using a different set of states, but still keeping the states as 2-dimensional. You will show that this does not help very much

Consider the unitary U defined over a 3-qubit system:

$$U = \frac{1}{\sqrt{6}} \begin{pmatrix} 2 & * & * & * & 0 & * & * & * \\ 0 & * & * & * & 0 & * & * & * \\ 0 & * & * & * & 1 & * & * & * \\ 1 & * & * & * & 0 & * & * & * \\ 0 & * & * & * & 1 & * & * & * \\ 1 & * & * & * & 0 & * & * & * \\ 0 & * & * & * & 0 & * & * & * \\ 0 & * & * & * & 2 & * & * & * \end{pmatrix}$$

The exact choice of $*$ doesn't matter, as long U is unitary. Notice that the rows and columns are indexed by triples of bits $\{000, 001, 010, 011, 100, 101, 110, 111\}$, so that the two specified columns correspond to 000 and 100.

Now consider *any* single-qubit state $|\psi\rangle$. Any such state can be written as $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ for complex numbers α, β such that $|\alpha|^2 + |\beta|^2 = 1$.

Consider applying U to $|\psi\rangle|0\rangle|0\rangle$, giving $U|\psi\rangle|0\rangle|0\rangle$, and measuring only the very last qubit, resulting in measurement outcome b. Let p_0 be the probability $b = 0$ and p_1 the probability $b = 1$. Then the state collapses to $|\phi_b\rangle|b\rangle$ for some quantum states $|\phi_b\rangle$.

Part (a). 5 points. Compute p_0, p_1 as functions of α, β.

Part (b). 10 points. Use the rules of partial measurements to write out expressions for $|\phi_0\rangle$ and $|\phi_1\rangle$ as functions of α, β.

Part (c). 10 points. We will now claim that the states $|\phi_0\rangle$ and $|\phi_1\rangle$ are "close" in some sense to $|\psi\rangle|\psi\rangle$. In particular, the probability that $|\phi_b\rangle$ is accepted as a clone of $|\psi\rangle$ is $|(\langle\psi|\langle\psi|)|\phi_b\rangle|^2$. The overall probability of success of this attack is therefore

$$P_{\mathsf{succ}} = p_0|(\langle\psi|\langle\psi|)|\phi_0\rangle|^2 + p_1|(\langle\psi|\langle\psi|)|\phi_1\rangle|^2$$

Show that $P_{\mathsf{succ}} = 2/3$, independent of α, β. *[Hint: You may fine this identity useful:* $(x+y)^3 = x^3 + 3x^2y + 3xy^2 + y^3$*].*

Remark 1. *That completes Problem 1. Note that you gave an attack that works with probability* $2/3$*. But in class, we claimed that Wiesner's scheme has an optimal attack probability of* $3/4$*. This is because Wiesner's scheme uses only real-values states, and it turns out that this allows for a slightly better attack. The following unitary gives an attack success probability of* $3/4$ *for any real-values states, showing that Wiesner's scheme is optimal among real-valued states.*

$$V = \frac{1}{\sqrt{12}} \begin{pmatrix} 3 & * & * & * & 0 & * & * & * \\ 0 & * & * & * & 1 & * & * & * \\ 0 & * & * & * & 1 & * & * & * \\ 1 & * & * & * & 0 & * & * & * \\ 0 & * & * & * & 1 & * & * & * \\ 1 & * & * & * & 0 & * & * & * \\ 1 & * & * & * & 0 & * & * & * \\ 0 & * & * & * & 3 & * & * & * \end{pmatrix}$$

2 Problem 2 (20 points)

Here, you will show that any efficient classical circuit can be simulated by an efficient quantum circuit. In particular, given a classical circuit C, you will construct a quantum circuit implementing the unitary $U_C|x, y\rangle = |x, y \oplus C(x)\rangle$.

A crucial ingredient is the Toffoli, also called the CCNOT gate. It is defined over three qubits as $\mathsf{CCNOT}|a, b, c\rangle = |a, b, c \oplus (ab)\rangle$. In matrix notation, it looks like:

$$\mathsf{CCNOT} = \begin{pmatrix} 1 & & & & & & & \\ & 1 & & & & & & \\ & & 1 & & & & & \\ & & & 1 & & & & \\ & & & & 1 & & & \\ & & & & & 1 & & \\ & & & & & & & 1 \\ & & & & & & 1 & \end{pmatrix}$$

Part (a). 10 points. Let C be any classical circuit made of AND, NOT gates (OR gates can be realized by AND and NOT gates). Let q be the total number of gates, n the number of input bits, and m the number of output bits.

Let $D_C(x)$ be the function that ouputs the values of *all* internal and output wires in $C(x)$ (there are q such wires, one for each gate in the circuit). Construct a quantum circuit for U_{D_C}. In particular, $U_{D_C}|x, 0^m, 0^{q-m}\rangle = |x, C(x), w(x)\rangle$, where $w(x)$ is the list of wire values for all internal wires in the circuit. H

You circuit will contain only CCNOT, CNOT, and NOT gates. The number of such gates is at most $O(q)$. Remember that, since this is a quantum circuit, wires cannot be branched, and all wires coming out of a gate must either go into another gate or be output wires.

[Hint: Suppose you have three qubits in the state $|a, b, 0\rangle$*. How can you put* $a\mathsf{AND}b$ *into the third qubit, using a* CCNOT *gate? How about putting* $\mathsf{NOT}(a) = 1 - a$ *into the third qubit, using a* CNOT*?*

Part (b). 5 points. Let $D(x)$ be a function, and consider a function $E(x) = g(x, D(x))$ for some function g. Explain how you can construct U_E using a constant number of evaluations to U_g and U_D, plus potentially a few CNOT gates, swapping a few wires, and a few ancilla registers which start at $|0^{q'}\rangle$ and are returned to $|0^{q'}\rangle$. *[Hint: Consider first applying* U_D*, and then* U_g *to the result. Some of the wires will now contain the result of* $E(x) = g(x, D(x))$*. But there will also be wires containing the value of* $D(x)$*. How might you return these wires to* 0*?]*

Part (c). 5 points. Explain how to take your implementation of U_{D_f} from Part (a), and turn it into U_f, using the result of Part (b).

Remark 2. *Above, used* CCNOT*, which is a 3-qubit gate. It turns out that it is possible to construct* CCNOT *with just a constant number of* CNOT *and 1-qubit gates. Thus, we can implement the circuits above using* CNOT *and 1-qubit gates with a constant-factor overhead.*

3 Problem 3 (30 points)

Consider a function $f : \{0,1\}^n \to \{0,1\}$ such that there are exactly K points such that $f(x) = 1$. In class, we saw how Grover's algorithm can find a random such point using $O(\sqrt{2^n/K})$ evaluations of f.[1]

Part (a). 15 points. Show that *all* K points can be found using $O(\sqrt{2^n K})$ evaluations of f. (note that K is in the numerator).

[Hint: One thing that doesn't quite work is to just run Grover's algorithm many times until all K distinct points are found. Since the points outputted by Grover's algorithm are random accepting points, eventually all points will be found. The number of tries necessary is known as the Coupon Collector's problem, and it is known that the number of trials needed is $\Theta(K \log K)$, which gives an overall run-time of $O(2^n K \log K)$. To shave of the log factor, you will only run Grover for K times, but need to ensure that all the K points are distinct. You will accomplish this by modifying running Grover on a modified function f', and each run of Grover will use a different f'. f' is derived from f, and you can implement $U_{f'}$ from U_f per Problem 2 Part (b). You will find the following useful: $\sum_{i=1}^{K} K^{-1/2} = O(K^{1/2})$.].

Part (b). 15 points. Let $g : \{0,1\}^n \to \{0,1\}^m$ be a function with the promise that, for each $x \in \{0,1\}^m$, there is exactly one $x' \neq x$ such that $g(x) = g(x')$. Your goal is to find such a pair (x, x').

To do so, first evaluate g on T random points, for some T that will be chosen later. Let L be the list of inputs queried. If L contains a pair $x \neq x'$ with $g(x) = g(x')$, then we are done.

Otherwise, use Grover's algorithm to find an $x' \notin L$ such that there exists $x \in L$ satisfying $g(x') = g(x)$. To do so, define a function f such that $f(x') = 1$ if and only if x' is such a point. f can be evaluated by making a call to g. Then run Grover's algorithm on f to find such an x'. Once you have x', you just look in the list L for the corresponding x, and output (x, x'). Note that f will have the evaluations $g(x)$ that you already computed for all $x \in L$ included in its description. How long does Grover's algorithm take on this f (in terms of the number of times is must evaluate g)? Call this time T', which will be a function of T and 2^n. (You may assume searching through a list of values takes constant time)

Finally, choose T to optimize the overall running-time. What is the overall running time of the algorithm? You only need to count the number of evaluations of U_g.

Remark 3. *Your algorithm will technically be calling U_f for f that are a modification of g. But per Problem 2 Part (b), we can implement U_f by making queries to U_g. We are only counting the total number of queries to U_g, since this will be the dominant cost.*

Remark 4. *The best classical algorithm for finding x, x' simply tries a number of random guesses until it finds one. This takes time $O(\sqrt{2^n})$ to find such a pair (each pair has a 2^{-n} probability of being having the same output, and with $O(\sqrt{2^n})$ queries you get $O(2^n)$ pairs). This is known as the birthday paradox.*

OPTIONAL BONUS: Part (c). 5 points. Suppose instead g is 3-to-1, meaning for every $x \in \{0,1\}^n$, there are exactly two inputs x', x'' such that $g(x) = g(x') = g(x'')$. Give a quantum algorithm for finding such a triple (x, x', x'') in time $O(2^{3n/7})$.

[1] Technically, this is only with overwhelming probability, but you may ignore that for this question.

Remark 5. *The best classical algorithm takes time $O(2^{2n/3})$: each triple has a probability 2^{-2n} of having the same outputs, and $O(2^{2n/3})$ queries gives a total of $O(2^{2n})$ triples. You will receive partial bonus points for any algorithm that does better than $O(2^{2n/3})$, with maximum points for $O(2^{3n/7})$, which turns out to be optimal.*

4 Problem 4 (10 points)

The hidden shift problem is closely related to the hidden subgroup/period finding problem. Here, we have some group $\mathbb{G}$ (for this problem, $\mathbb{G}$ is abelian and written additively), and two functions $f_0, f_1 : \mathbb{G} \to \{0,1\}^*$. The promise is that $f_1(x) = f_0(x+s)$ for some hidden $s \in \mathbb{G}$. The goal is to find s.

Part (a). 5 points. Show that if $\mathbb{G} = \mathbb{Z}_2^n$, then the hidden shift problem can be solved in polynomial time. You may use period-finding (Simon's or Shor's algorithms) as a black box.

Part (b). 5 points. Now consider the case $\mathbb{G} \neq \mathbb{Z}_2^n$ for any n. One might hope that, as with the hidden subgroup/period-finding problem, hidden shift can still be solved in quantum polynomial time. However, this appears to be false. Explain why there is no obvious way to use period finding to solve Hidden Shift in this case, despite your answer to Part (a).

5 Problem 5 (15 points)

The Evan-Mansour cipher takes a *public* permutation $P : \{0,1\}^n \to \{0,1\}^n$, and turns it into a private-key encryption scheme. The key is a pair $(k_0, k_0) \in (\{0,1\}^n)^2$, and the encryption of a message m is $\mathsf{Enc}((k_0,k_0),m) = k_0 \oplus P(k_0 \oplus m)$. Decryption is straightforward using P^{-1}.

The construction cannot be CPA secure (do you remember why?). However, it could plausibly still have some weaker form of security. In this problem, you will show that if *quantum* queries are made to $\mathsf{Enc}((k_0,k_1),\cdot)$, then it is possible to actually recover the key k_0, k_1, showing that no security is possible under such queries.

Part (a). 10 points. Suppose you are given quantum oracle access to the unitary U_g where $g(m) = \mathsf{Enc}((k_0,k_1),m)$ for unknown keys k_0, k_1. Construct a function $f(m)$ which uses g and P as a sub-routine, with the guarantee that $f(m \oplus k_0) = f(m)$.

[Hint: Think first about the case $k_1 = 0^n$, and compare $\mathsf{Enc}((k_0,k_1),m)$ to $P(m)$.]

Thus, running Simon's algorithm on f will give k_0. Complete the attack by finding k_1 once k_0 is known. Note that technically, Simon's algorithm required $f(x) \neq f(y)$ for $y \notin \{x, x \oplus k_0)$, but we will ignore this issue since this will be true for "most" such x, y.

Part (b). 5 points. A simple fix to the above is to replace $+$ with addition modulo 2^n, interpreting strings $\{0,1\}^n$ as integers $[0, 1, \cdots, 2^n - 1]$ in the natural way. Explain why Simon's and Shor's algorithms don't work straightforwardly with this change.