Solutions to Problem Set 3

1. Let $G : \{0,1\}^n \rightarrow \{0,1\}^{2n}$ be a $(t, \epsilon)$-secure pseudorandom generator computable in time $r$.

Show that $G$ is also a $(t - r - O(n), \epsilon + 2^{-n})$-secure one way function.

Solution. Suppose that $A$ is an algorithm of complexity $t - r - O(n)$ such that

$$\mathbb{P}_x [A(G(x)) = x' : G(x) = G(x')] > \epsilon + 2^{-n} \quad (1)$$

Consider the algorithm $A'$ that, on input $y$, computes $A(y)$ and then outputs 1 if and only if $G(A(y)) = y$. Then, from (1) we have

$$\mathbb{P}_{x \in \{0,1\}^n} [A'(G(x)) = 1] > \epsilon + 2^{-n}$$

Now note that we can have $A'(y) = 1$ only if $y$ is a possible output of $G$, and $G$ has at most $2^n$ possible outputs so

$$\mathbb{P}_{z \in \{0,1\}^{2n}} [A'(z) = 1] \leq \mathbb{P}_{z \in \{0,1\}^{2n}} [z \text{ is a possible output of } G] \leq \frac{2^n}{2^{2n}} = 2^{-n}$$

and so

$$\left| \mathbb{P}_{x \in \{0,1\}^n} [A'(G(x)) = 1] - \mathbb{P}_{z \in \{0,1\}^{2n}} [A'(z) = 1] \right| > \epsilon$$

and $A'$ has complexity $\leq t$, thus contradicting the $(t, \epsilon)$ pseudorandomness of $G$.

2. Let $f : \{0,1\}^n \rightarrow \{0,1\}^m$ be a $(t, \epsilon)$-secure one-way function.

Show that

$$\frac{t}{\epsilon} \leq O((m + n) \cdot 2^n)$$
Solution. We need to show that for every $\epsilon$ there is an algorithm $A_\epsilon$ of complexity $\leq O((m + n) \cdot \epsilon \cdot 2^n)$ such that

$$\Pr_x[A_\epsilon(f(x)) = x' : f(x) = f(x')] \geq \epsilon$$

We define $A_\epsilon$ to have a look-up table contained $\epsilon \cdot 2^n$ pairs $(x, f(x))$, one for each $x$ belonging to an arbitrarily chosen set $S$ of size $\epsilon 2^n$. (For example the first $\epsilon 2^n$ strings in lexicographic order.)

On input $y$, we determine if $y$ is a second element of any pair in the table, and, if so, we output the first element. If the look-up table is sorted, the algorithm can use binary search, and have running time $O((m + n) \cdot n)$; the size of the table dominates the complexity.

3. Let $f : \{0, 1\}^n \rightarrow \{0, 1\}^n$ be a $(t, \epsilon)$-secure one-way permutation computable in time $\leq r$.

Show that

$$\frac{t^2}{\epsilon} \leq O((r + n^2)^2 \cdot 2^n)$$

[Hint: first show that, for any permutation $f : \{0, 1\}^n \rightarrow \{0, 1\}^n$, there is an algorithm of complexity $O(r \cdot 2^{n/2})$ that inverts the permutation everywhere. The algorithm is given a pre-computed data structure of size $O(n 2^{n/2})$ and runs in time $O(r 2^{n/2})$. Recall that in our model of computation we do not pay for the price of pre-computing data at “compile time,” we only pay the sum of the length of the program, including any fixed data it needs access to, plus the worst-case running time.]

Solution. We need to show that for every $\epsilon$ there is an algorithm $A_\epsilon$ of complexity $O((r + n^2)^2 \sqrt{\epsilon} 2^n)$ that inverts $f()$ on at least an $\epsilon$ fraction of inputs.

Consider the graph that has one vertex for every element $x \in \{0, 1\}^n$ and one directed edge $(x, f(x))$ for every $x \in \{0, 1\}^n$. Thus, every vertex has in-degree one and out-degree one, and the graph is a collection of disjoint cycles. The problem of inverting $f()$ can be thought of as the problem: given a vertex in the graph, find the predecessor of that vertex in the cycle that it belongs to.

A simple algorithm for inverting $f()$ is to simply “walk” on the graph: given $y \in \{0, 1\}^n$, we compute $f(y), f(f(y))$, and so on, until we return to the value $y$; the value we encounter before returning to $y$ is the unique $x$ such that $y = f(x)$. Unfortunately, if $f()$ defines a graph containing just one huge cycle, then the running time of this algorithm is $r \cdot 2^n$, which is no better than trying all possible pre-images by brute force.
The idea is then to construct a data structure containing “shortcuts.” Let
\( \ell = \sqrt{\epsilon 2^n} \) (assume for simplicity that it’s an integer). For every cycle of length
\( L \geq 2\ell \), we pick vertices \( x_1, \ldots, x_k \), \( k = \lfloor L/\ell \rfloor \), which are equally spaced around
the cycle (possibly \( x_k \) is slightly closer to \( x_1 \) to account for rounding error), and
we add the pairs \((x_1, x_2), (x_2, x_3), \ldots, (x_k, x_1)\) to the data structure. Note that
at most a \( 1/\ell \) fraction of vertices in each cycle give rise to elements of the data
structure. We stop the construction when we run out of long cycles or when we
have added \( \sqrt{\epsilon 2^n} \) pairs to the data structure, whichever comes first.

Now consider the algorithm \( A_\epsilon \)

- Input: \( y \)
- \( y_0 := y \)
- for \( i := 0 \) to \( 2\sqrt{\epsilon 2^n} \)
  - If there is a pair \((x_1, x_2)\) in the data structure such that \( x_2 == y_i \),
    then \( y_{i+1} := x_1 \)
  - Else \( y_{i+1} := f(y_i) \)
  - If \( y_{i+1} == y \) then return \( y_i \)
- return \( FAIL \)

that, on input \( y \), computes \( y_1 = f(y) \), \( y_2 = f(y_1) = f(f(y)) \) and so on as
before, but, in addition, at every step checks not only whether \( y_{i+1} = y \), but
also whether \( y_i \) is a second element of a pair in the data structure. In the first
case, of course we output \( y_i \). In the second case, we continue with the first
element of the pair, which corresponds to moving backwards on the cycle by
roughly \( \ell \) steps (and no more than \( 2\ell \)). If we don’t find an inverse within \( 2\ell \)
steps, we fail.

Note that we invert all the elements that belong to cycles of length \( \leq 2\ell \), and,
for every pair that we add to the data structure, we add at least \( \ell \) elements to
the set of inputs that are correctly inverted by the algorithm, so the algorithm
correctly inverts at least \( \ell^2 \) elements, that is, at least \( \epsilon 2^n \), which is at least an
\( \epsilon \) fraction of the total.

Every step of the algorithm requires time \( r \) to evaluate \( f \) and time \( O(n^2) \) to
search the data structure.

4. Let \( f : \{0,1\}^n \rightarrow \{0,1\}^n \) be a \((t,\epsilon)\)-secure one-way function computable in time
\( r \).

Show that \( g : \{0,1\}^{2n} \rightarrow \{0,1\}^{2n} \) defined as \( g(x, y) := f(x), f(y) \) is \((t-O(r),\epsilon)\)-
secure.
Solution. Suppose that $g$ is not $(t - O(r), \epsilon)$ secure, and let $A$ be an algorithm of complexity $t_A = t - O(r)$ such that

$$\mathbb{P}_{x,y}[A(f(x), f(y)) = (x', y') : f(x') = f(x) \land f(y') = f(y)] > \epsilon$$

Then consider the algorithm $A'$ that, on input $z$, picks a random $y \in \{0, 1\}^n$ and simulates $A(z, f(y))$, then return the first output of $A(z, f(y))$.

We have

$$\mathbb{P}_x[A'(f(x)) = x' : f(x) = f(x')] = \mathbb{P}_{x,y}[A(f(x), f(y)) = (x', y') : f(x') = f(x)] > \epsilon$$

And note that $A'$ has complexity $\leq t_A + n + r \leq t$, contradicting the $(t, \epsilon)$ security of $f$.

5. Let $p$ be a prime and $g$ be a generator for $\mathbb{Z}_p^*$ such that $f(x) := g^x \pmod{p}$ is a $(t, 0.99)$-one way permutation. Let $k = \lceil \log_2 p \rceil$ be the number of digits of $p$; then recall that $f()$ is computable in time $O(k^3)$.

Show that $f()$ is also $(\frac{1}{24,000}(t - O(k^3)), 0.51)$-one way.

Solution. Suppose $f()$ is not $(\frac{1}{24,000}(t - O(k^3)), 0.51)$ one way, so that there is an algorithm $A$ of complexity $t_A \leq \frac{1}{24,000}(t - O(k^3))$ such that

$$\mathbb{P}_{x \in \{0, \ldots, p-1\}}[A(g^x) = x] \geq 0.51$$

Now, suppose we are given $y = g^x$, and consider the process of picking a random $r \in \{0, \ldots, p - 1\}$ and computing $A(y \cdot g^r)$. Then, with probability at least .51, the answer will be the correct one, $x + r \pmod{p}$. If we subtract $r$, we get $x$. This process succeeds for every $x$, with probability at least 51%. If we repeat the process for $k$ randomly chosen $r$, then we will find the correct answer on average at least .51 · $k$ times, and the probability that the majority answer is correct is at least

$$1 - e^{-2k/10,000}$$

using the Chernoff bound. This probability is at least .99 provided that $k \geq 23026$.

Overall, we have the following algorithm $A'$:
• Input: \( y \)
• For \( i = 1 \) to 23,026
  - Pick random \( r \)
  - \( x_i := A(g^r \cdot y) - r \mod (p - 1) \)
• Return most frequent value among \( x_i \)

This algorithm runs in time \( \leq 23,026 \cdot (t_A + O(k^3)) \leq t \) and inverts \( f \) on more than a 99% fraction of inputs.

6. Recall that if \( F : \{0, 1\}^n \rightarrow \{0, 1\}^n \) is a function then the we define the Feistel permutation \( D_F(x, y) := y, x \oplus F(y) \).

Show that there is an efficient oracle algorithm \( A \) such that

\[
P_{\Pi:\{0,1\}^{2m} \rightarrow \{0,1\}^{2m}}[A^{\Pi,\Pi^{-1}} = 1] = 2^{-\Omega(m)}
\]

where \( \Pi \) is a random permutation, but for every three functions \( F_1, F_2, F_3 \), if we define \( P(x) := D_{F_3}(D_{F_2}(D_{F_1}(x))) \) we have

\[
A^{P,P^{-1}} = 1
\]

[Note: I don’t know the solution.]