1 Fun with PRGs

Let $G_1, G_2 : \{0, 1\}^n \rightarrow \{0, 1\}^{2n}$ and $G_3 : \{0, 1\}^n \rightarrow \{0, 1\}^{3n}$ be length-doubling and length-tripling PRGs, respectively, for every $n$. Let $s, s_1, s_2 \in \{0, 1\}^k$. Let $|x|$ denote the length of the binary string $x$.

For each of the following, either prove that it is a PRG or provide a counterexample to show that it is not necessarily a PRG.

a. $G_a(s) = G_1(s) \oplus G_2(s)$

b. $G_b(s_1 \circ s_2) = G_3(s_1) \oplus G_3(s_2)$.

Recall that $(\circ)$ is the parse operator that denotes dividing $s$ into two equal-length strings $s_1$ and $s_2$. If $s$ is of odd length, then $(\circ)$ will split $s$ into $s_1$ and $s_2$ of equal length, ignoring the last bit of $s$. Hence we have that $|s_1| = |s_2| = \left\lfloor \frac{|s|}{2} \right\rfloor$.

2 Pseudorandom Generators and the Hybrid Argument

Recall the construction of a pseudorandom generator from a one-way permutation family $f_{pk}$ with a hardcore bit $B_{pk}$ presented in class. The construction proceeds as follows: On input a $k$-bit seed $s$, we want to output $2k$ random-looking bits. Set $s_0 = s$ and $s_i = f_{pk}(s_{i-1})$ for $1 \leq i \leq k$. Finally, output the string $s_k b_1 b_2 \cdots b_k$, where $b_i = B_{pk}(s_{i-1})$. By $\text{construction}(pk, s)$, we denote the final string $s_k b_1 b_2 \cdots b_k$ that the construction yields given the random seed $s$.

To prove the security of the construction, we introduce a series of hybrid distributions:

$$H_i(1^k) = \{pk \leftarrow \text{KeyGen}(1^k);$$

$s \leftarrow \{0, 1\}^k$;

$r_1 \cdots r_i \leftarrow \{0, 1\}^i$;

$s_k b_1 b_2 \cdots b_k \leftarrow \text{construction}(pk, s):$

$(s_k r_1 \cdots r_i b_{i+1} \cdots b_k, pk)$

where $0 \leq i \leq k$. 
Lemma 1  If $B_{pk}$ is a hardcore bit of the one-way permutation $f_{pk}$, then for all p.p.t. algorithms $A$, there exists a negligible $\nu$ such that for all $k$ where $0 \leq i \leq k - 1$, $A$’s advantage in distinguishing $H_i(1^k)$ from $H_{i+1}(1^k)$ is at most $\nu(k)$.

Prove this lemma.

Hint: The proof will require a reduction: Given an adversary $A$ distinguishing $H_i$ from $H_{i+1}$ with nonnegligible advantage, construct an algorithm $B$ that contradicts some assumption. Which assumption? What will $B$ receive as input, and what should its output be? What should it feed $A$ and how should it interpret $A$’s output? Give the reduction and analyze it.

3  Decisional Diffie-Hellman Assumption and PRGs

Let $G$ be a cyclic group of prime order $q$ with generator $g$. Recall that a group is cyclic if it can be generated by a single element, which is called a generator. Hence each element $y \in G$ is equal to $g^x$ for some $x \in \mathbb{Z}_q$. The decisional Diffie-Hellman (DDH) assumption states that for for $a,b,c \leftarrow \mathbb{Z}_q$ drawn uniformly at random, the following two distributions are computationally indistinguishable:

$$(g, g^a, g^b, g^{ab}) \approx (g, g^a, g^b, g^c)$$

a. Let $w = p(\log q)$ where $p$ is some polynomial. Prove that under the DDH assumption, the following two distributions are indistinguishable:

$$(g^{a_1}, g^{a_2}, \ldots, g^{aw}, g^{a_1b}, g^{a_2b}, \ldots, g^{awb}) \approx (g^{a_1}, g^{a_2}, \ldots, g^{aw}, g^{c_1}, g^{c_2}, \ldots, g^{cw})$$

where $a_i, b, c_i \leftarrow \mathbb{Z}_q$ are all uniformly random and independent.

b. Using what you proved in part (a), construct a PRG $G : \{0,1\}^n \rightarrow \{0,1\}^m$. What would you choose as your $m,n$ and what would be your seed? Assume that each of the $q$ elements in $G$ can be represented by binary strings of length $\lceil \log q \rceil$.

Prove the security of your $G$ with the help of a reduction argument to the DDH assumption.

c. Let $p = 2q + 1$ also be a prime. Prove that DDH assumption does not hold in $\mathbb{Z}_p^*$. 
4 Random Self-Reducibility of RSA

In this problem, when talking about multiplication of elements in $\mathbb{Z}_N^*$, we omit the implicit mod $N$.

a. Prove that all of the following distributions are identical for all RSA public keys $(N, e)$ and any $y \in \mathbb{Z}_N^*$:

$$
D_1 = \{r \leftarrow \mathbb{Z}_N^* : r\}
$$

$$
D_2 = \{r \leftarrow \mathbb{Z}_N^* : r^e\}
$$

$$
D_3 = \{r \leftarrow \mathbb{Z}_N^* : r^ey\}
$$

By identical distributions, we mean that for all $x \in D_1 \cup D_2 \cup D_3$, we have $p_1(x) = p_2(x) = p_3(x)$, where $p_i(x) = \Pr[\hat{x} \leftarrow D_i : \hat{x} = x]$ for $i \in \{1, 2, 3\}$.

b. Suppose you are given an RSA public key $(N, e)$ and the values $y, r, z \in \mathbb{Z}_N^*$ such that $z^e = r^ey$. How do you compute $x$ such that $x^e = y$?

c. Suppose you are given an algorithm $A$ and an RSA public key $(N, e)$ such that for some $\varepsilon > 0$,

$$
\Pr[r \leftarrow \mathbb{Z}_N^*; z \leftarrow A(N, e, r) : z^e = r] = \varepsilon
$$

In other words, when given as input a random element of $\mathbb{Z}_N^*$, $A$ outputs its $e$th root with probability $\varepsilon$. Design an algorithm $B$ such that for all $y \in \mathbb{Z}_N^*$,

$$
\Pr[x \leftarrow B(N, e, y) : x^e = y] = \varepsilon
$$

Your algorithm $B$ must run $A$ as a subroutine. In other words, your algorithm must take its input $y$ and transform it into a random-looking $r$, run $A(N, e, r)$, obtain $z$ such that $z^e = r$, and from $z$, determine $x$.

d. The RSA assumption is that, given an RSA public key $(N, e) \leftarrow \text{KeyGen}_{\text{RSA}}(1^k)$ and a random $y \in \mathbb{Z}_N^*$, it is hard to find $x \in \mathbb{Z}_N^*$ such that $x^e = y$. In experiment notation: For all p.p.t. adversaries $A$, there exists a negligible function $\nu(k)$ such that:

$$
\Pr[(N, e) \leftarrow \text{KeyGen}_{\text{RSA}}(1^k); y \leftarrow \mathbb{Z}_N^*; x \leftarrow A(N, e, y) : y = x^e] = \nu(k)
$$

Show that the following assumption is equivalent to the RSA assumption (i.e. it holds if and only if the RSA assumption is true): For all p.p.t. adversaries $A$, there exists a negligible function $\nu(k)$ such that:

$$
\Pr[(N, e) \leftarrow \text{KeyGen}_{\text{RSA}}(1^k); y \leftarrow \text{QR}_N; x \leftarrow A(N, e, y) : y = x^e] = \nu(k)
$$

**Hint:** Use reductions.