Problem 1

The DSA (Digital Signature Algorithm) was proposed by the National Institute of Standards and Technology (NIST) in August 1991 for use in their Digital Signature Standard (DSS). In this problem, we try to break the DSA under various attacks. For starters, the DSA works as follows:

- **Key-Generation:** $G(1^n)$. Pick primes $p,q$ such that $p = 2q + 1$. Pick a generator $h$ of the multiplicative group $\mathbb{Z}_p^*$, and set $g = h^2 \pmod{p}$. (Thus, $g$ is a generator of a subgroup of order $q$ of $\mathbb{Z}_p^*$.) Pick a random element $x \in \mathbb{Z}_p^*$. The public (verification) key is $(p, g, gx)$, and the secret (signing) key is $x$.

- **To sign a message** $m$: Choose $k$ at random in $\mathbb{Z}_q^*$. Let $r = g^k \pmod{p}$. Solve the equation $rx - ks \equiv m \pmod{q}$ to find the unique solution $s$. Output $(r, s)$. (Note: $k$ is kept secret.)

- **To verify a signature** $(r, s)$ of a message $m$: Check if $(g^x)^r r^{-s} \equiv g^m \pmod{p}$. If this equation checks, then output “signature verified” else output “fail”.

(Before starting on the problems, convince yourself that this is indeed a correct signature scheme; that is, that valid signatures will pass verification.)

(a) Why does the $k$ need to be secret in the signing algorithm? Find a way to recover the secret signing key, given a single signature $(r, s)$ of a message $m$, along with the randomness $k$ used to generate the signature.

**Solution:** We know that $rx - ks \equiv m \pmod{q}$. If $k$ is not kept secret, then since $q$ is public and we have a message $m$ and corresponding signature $(r, s)$, we can simply solve the linear equation for $x$ to obtain the secret key.

(b) Usually (in practice), the $k$ used for signing is not generated completely at random, but by using a pseudorandom generator. Again, in practice, people use a simple “pseudorandom” generator such as the linear congruential generator. A linear congruential generator $LCG$, on input a seed $s_0$ outputs a sequence $(s_1, s_2, \ldots, s_n)$ such that $s_i = as_{i-1} + b \pmod{q}$. (where $a$ and $b$ are assumed to be public parameters).

Show that if the $k$’s are generated using an $LCG$, then given two signatures (of two messages), one can recover the secret signing key. ("Lesson": Beware of using unproven, patched-up pseudorandom generators).

**Solution:** Note that if $s_i = as_{i-1} + b \pmod{q}$, then any two $s$ values are related by a linear equation modulo $q$. Namely, for $j > i$, $s_j = a^{j-i} s_i + b (a^{j-i-1} + a^{j-i-2} + \ldots + 1) \pmod{q} = a' s_i + b' \pmod{q}$. Note that since $a$ and $b$ are known, the values of $a'$ and $b'$ are depend only on $(j - i)$, the difference in indices.
Thus, we have:

\[ rx - ks \equiv m \pmod{q} \]
\[ r'x - ks' \equiv m' \pmod{q} \]
\[ s' \equiv a's + b' \pmod{q} \]

We can solve this system of three linear equations in three unknowns \((x, s, s')\), assuming we know the difference in indices between the two messages. (If we don’t, we can test each possible candidate \((j - i)\) and come up with a candidate \(x\) for that difference, which gives us a nonnegligible advantage in guessing \(x\) since the signer can only calculate polynomially many iterations of the LCG.)

(c) Finally, assume that the \(k\)'s are generated by a cryptographically strong pseudorandom generator (e.g. start with a truly random seed, and apply \(G\) to get a pseudorandom value twice as long as the \(k\) values. Use the first half as \(k_1\) and the second half as the next seed.). Prove that the DSA remains secure.

(In other words, show that if there is an adversary \(A\) that breaks the DSA that uses a PRG, then there is an adversary \(B\) that uses the original DSA – one where \(k\)'s are generated randomly.)

**Solution:** Assume \(A\) has a non-negligible chance of forging a signature under an instantiation of the DSA that uses a CS-PRG. We will show that \(A\) also has a non-negligible chance of forging a signature under an instantiation of the DSA that uses truly random \(k\) values.

Assume it does not. Then we create algorithm \(A'\) that can distinguish outputs of the PRG from random. \(A'\) works as follows: it takes in samples \(x\) that are either taken from a random distribution or the output of a PRG. It sets up the DSA system, acting as signer to adversary \(A\).

Whenever \(A\) requests a signature on a message, \(A'\) uses the next input sample as the randomness \(k\) used to generate the signature. If \(A\) eventually outputs a valid forgery, \(A'\) concludes that the samples were pseudorandom; if not, \(A'\) concludes that the samples were random.

By assumption, if the samples are pseudorandom, then \(A\) has a non-negligible chance \(\epsilon\) of outputting a forgery, whereas \(A\) only has a negligible chance \(\nu\) of forging if the samples are random. Thus, the probability \(A'\) is correct is

\[
P[\text{correct}] = P[\text{forges}|\text{pseudorandom}] \cdot P[\text{pseudorandom}] + P[\text{fails}|\text{random}] \cdot P[\text{random}] \\
\geq (1/2 + \epsilon)(1/2) + (1/2 - \nu)(1/2) \\
= 1/2 + (\epsilon - \nu)/2 \\
\]

which is a nonnegligible advantage over 1/2.

However, this contradicts the security of the pseudorandom generator. Thus, since the generator is secure by assumption, \(A\) must also have a nonnegligible chance of forging a message if the DSA uses random \(k\) values.

**Problem 2**

Consider the following interactive commitment scheme based on any pseudorandom generator \(G : \{0,1\}^n \rightarrow \{0,1\}^{3n}\) (which, recall, exists based on any one-way function). In this description, Alice plays the role of the sender (the person committing), and Bob is the receiver.

Let \(b \in \{0,1\}\) be the input of Alice.

**Commitment Phase:**
Bob selects a random string \( R \leftarrow \{0,1\}^{3n} \) and sends \( R \) to Alice.

Alice samples a random seed \( s \leftarrow \{0,1\}^n \) for the PRG. If \( b = 0 \), Alice sets \( C = G(s) \); if \( b = 1 \), Alice sets \( C = G(s) \oplus R \). Alice sends \( C \) to Bob.

**Reconstruction Phase:**

To decommit, Alice sends her input \( b \) together with the seed \( s \) to Bob, who accepts if the previously received commitment message \( C \) is equal to \( G(s) \oplus b \cdot R \).

(a) Prove that the scheme is *computationally hiding*. That is, at the conclusion of the commitment phase, no PPT malicious Bob can guess with non-negligible advantage (over the random coins of Alice) whether Alice committed to 0 or 1.

**Solution:** Suppose, to the contrary, there exists a malicious Bob who participates in the protocol with Alice and then can distinguish commitments to 0 from commitments to 1 with nonnegligible advantage \( \alpha \). We use this Bob to construct a new adversary \( D \), who distinguishes the output of the pseudorandom generator from truly random.

The distinguisher \( D \):
1. Run Bob for 1 step, and receive a message \( R \).
2. Flip a random bit \( b \). This will correspond to the bit that \( D \) will "commit" to.
3. Given challenge \( y \) from the PRG distinguishing game (i.e., \( y \) is either \( G(s) \) for some random \( s \) or is completely uniform), \( D \) sends \( y \oplus b \cdot R \) to Bob.
4. Bob outputs his prediction \( b' \) as to whether the commitment was to 0 or to 1. If \( b' = b \), then \( D \) guesses that \( y \) is pseudorandom. Otherwise, \( D \) guesses that \( y \) is random.

We now analyze \( D \)'s advantage in the PRG distinguishing game. If \( y \) is pseudorandom, then \( D \)'s message \( y \oplus b \cdot R \) is distributed exactly as a commitment to the bit \( b \), and thus \( D \) will identify the correct bit \( b \) with advantage corresponding to Bob's advantage \( \alpha \). However, if \( y \) is random, then the distribution of \( \{y \oplus b \cdot R\} \) is uniformly random, independent of the value of \( b \). Then, even a computationally unbounded Bob will not be able to identify the value of \( b \) given a sample from \( \{y \oplus b \cdot R\} \) with probability greater than \( \frac{1}{2} \). Thus, the probability that \( D \) outputs 1 (guess for pseudorandom) given \( y \) is pseudorandom will be non-negligibly higher than the probability that \( D \) outputs 1 given \( y \) is random.

(b) Prove that the scheme is *statistically binding*. That is, for all adversarial Alice (not necessarily computationally bounded), it holds in the reconstruction phase that

\[
\Pr_{Bob's \; coins} \left[ \text{Alice outputs} \ (s,s') \ \text{s.t.} \ \text{Bob accepts both} \ (0,s) \ \text{and} \ (1,s') \ \right] < \epsilon(n),
\]

for some negligible function \( \epsilon \).

**Solution:** We show that, over randomly chosen \( R \leftarrow \{0,1\}^n \), the probability that there *exists* a pair of seeds \( (s,s') \) such that \( G(s) = G(s') \oplus R \) is negligible (equivalently, \( G(s) \oplus G(s') = R \)). This holds because the range of the pseudorandom generator can be no larger than \( 2^n \) (corresponding to the \( 2^n \) different input seeds), which means there are at most \( 2^{2n} \) distinct values for \( G(s) \oplus G(s') \). However, \( R \) can take any of the \( 2^{3n} \) values in \( \{0,1\}^{3n} \). Thus, the probability for a randomly chosen \( R \) that it falls within the set of \( 2^{2n} \) possible \( G(s) \oplus G(s') \) values is bounded by \( 2^{-n} \).
Problem 3

Consider the language \( L = \{ (p, g) : p \in \text{PRIMES} \land g \in \text{GEN}(\mathbb{Z}_p^*) \} \). Without using any complexity assumption, prove that there exists a zero-knowledge proof system for the language \( L \).

**Solution:** A necessary condition for an instance \((p, g)\) to be in \( L \) is that \( p \) is a prime and \( p, g \in \mathbb{Z}_p^* \). Hence, without loss of generality, we consider only instances \((p, g)\) where \( p \) is a prime and \( p, g \in \mathbb{Z}_p^* \), because a verifier can efficiently check for that (and reject if the condition does not hold).

Consider the proof system \((P, V)\) such that, on common input \((p, g)\), the interaction is as follows:

1. The verifier \( V \) draws \( x \in \mathbb{Z}_p^* \) at random and sends \( x \) to the prover \( P \).
2. The prover \( P \) draws \( y \in \mathbb{Z}_p^* \) at random and sends \( y \) to the verifier \( V \).
3. The verifier \( V \) draws a bit \( b \) at random and sends \( b \) to the prover \( P \).
4. The prover sends \( z = \log_g(x^b y) \) to the verifier \( V \).
5. The verifier \( V \) accepts if and only if \( g^z \equiv x^b y \mod p \).

We claim that \((P, V)\) is a zero-knowledge proof system for \( L \). Thus, we need to show that \((P, V)\) satisfies completeness, soundness, and zero-knowledge.

First, we argue that \((P, V)\) is complete. For any \((p, g) \in L\), if both \( P \) and \( V \) follow the steps outlined above, then \( \Pr[PV(x) = 1] = 1 \), because \( g \) is a generator and \( P \) can compute \( \log_g \) modulo \( p \) for any element in \( \mathbb{Z}_p^* \) (in particular, it can compute \( \log_g(x^b y) \)).

Next, we argue that \((P, V)\) is sound. Let \((p, g)\) be not in \( L \), so that \( g \) is not a generator of \( \mathbb{Z}_p^* \). Let \( P' \) be any (not necessarily efficient) prover strategy. We show that \( \Pr[P'V(x) = \text{YES}] \leq 3/4 \) (and by repeating sequentially the protocol three times we reduce the probability that \( V \) output \( \text{YES} \) to less than \( 1/2 \)).

Since \( g \) is not a generator of \( \mathbb{Z}_p^* \), the order of \( g \) must be a non-trivial divisor of \( p - 1 \); in particular, the order of \( g \) is at most \( (p - 1)/2 \), and so at least half of the elements of \( \mathbb{Z}_p^* \) have no discrete logarithm with respect to base \( g \). Now, the verifier \( V \) chooses \( x \in \mathbb{Z}_p^* \) at random, so that the probability that \( x \) has a discrete logarithm with respect to base \( g \) is at most \( 1/2 \). Moreover, for any \( y \in \mathbb{Z}_p^* \), both \( y \) and \( x y \) have a discrete logarithm with respect to base \( g \) only if \( x \) does. Hence, with probability at least \( 1/2 \), at least one of \( y \) or \( x y \) does not have a discrete logarithm with respect to base \( g \). When this happens, with another probability \( 1/2 \), \( V \) chooses a bit \( b \) such that \( P' \) can’t give the discrete logarithm that \( V \) asks for, and \( V \) will reject. Thus, with probability at least \( 1/4 \), \( P' \) will get caught cheating, i.e., the probability that \( V \) accepts is at most \( 3/4 \).

Finally, we argue that \((P, V)\) is zero knowledge. Given any probabilistic polynomial-time verifier \( V' \), we exhibit a probabilistic polynomial-time simulator \( S \) such that, on input \((p, g) \in L\), we have \( S(p, g) = PV'[p, g] \) (where \( PV'[p, g] \) denotes the transcript of the communication between \( P \) and \( V' \) on the given input \((p, g)\), together with the private coin tosses used by \( V' \)); the simulator \( S \) will use \( V' \) as a (black-box) subroutine. We define \( S \) as the following algorithm:

0. Fill the random tape of \( V' \) with truly random coins \( r \).
1. Run \( V' \) on input \((p, g) \) and \( r \), and get \( x \) from \( V' \). (If \( V' \) halts instead, output \( r \) and halt. If \( x \notin \mathbb{Z}_p^* \), output \((x, r) \) and halt.)
2. Draw $z$ in $\mathbb{Z}_p^*$ at random.
3. Draw a bit $b^*$ at random.
4. Compute $y \equiv g^z (x^{-1})^{b^*} \mod p$.
5. Give $y$ to $V'$, and get $b$ from $V'$. (If $V'$ halts instead, output $((x, y), r)$ and halt. If $b$ is not a bit, output $((x, y, b), r)$ and halt.)
6. If $b = b^*$, output $((x, y, b, z), r)$ and halt. Otherwise go to Step 1 and repeat.

When $(p, g) \in L$, no matter what $b^*$ and $x$ are, $y$ is a random element in $\mathbb{Z}_p^*$, because $g$ is a generator and $z$ is a random element in $\mathbb{Z}_p^*$. Hence, $V'$ has no idea about $b^*$ when we give $y$ to him, and, with probability $1/2$, $b = b^*$. Thus the expected running time of $S'$ is polynomial in $|p| + |g|$. Moreover, when $b = b^*$, $z$ is the discrete logarithm base $g$ of $x^by$. Therefore, $((x, y, b, z), r)$ follows the correct distribution, as desired. (To simulate the view of $V'$ when the protocol is repeated three times, as required by soundness, $S$ simply does the simulation round by round, fixing the random coins used by $V'$ once and for all, and then simulating each round, and starting from scratch with the same random tape if $S$ fails.)

**Problem 4**

Given a language $L$, for each $k \in \mathbb{N}$ define the two languages $L_k$ and $\overline{L}_k$ as follows:

$$L_k = \{ x : x \in L \land |x| = k \},$$

$$\overline{L}_k = \{ x : x \notin L \land |x| = k \}.$$

Suppose that $L$ has a zero-knowledge proof system $(P, V)$. Moreover, suppose that some (possibly cheating) verifier $\tilde{V}$ can use his view to convince a third party, i.e., there exists a probabilistic polynomial-time algorithm $D_L$ and a positive constant $c_L$ such that for all sufficiently large $k$ the following conditions hold:

1. For all $x \in L_k$,
   $$\Pr \left[ D_L(x, P \tilde{V}[x]) = 1 \right] > \frac{1}{2} + \frac{1}{k^{c_L}}.$$
2. For all $x \in \overline{L}_k$ and for all probabilistic polynomial-time algorithms $B$,
   $$\Pr \left[ D_L(x, B(x)) = 1 \right] < \frac{1}{2} - \frac{1}{k^{c_L}}.$$

Prove that $L \in \text{BPP}$.

**Remark.** This problem answers the question of whether there is a possibly cheating verifier $\tilde{V}$ that, after communicating with the prover $P$, is able to use his own view for $x \in L$ to convince a third party (who was not present when $P$ and $\tilde{V}$ communicated) that $x \in L$. In particular, this says that, if $L \notin \text{BPP}$, then, even if $x \notin L$, $\tilde{V}$ may still be able to generate a fake view for $x \in L$ that looks good to any third party; because of this, $\tilde{V}$ will not be able to use his real view for $x$ to convince anybody else.

**Solution:** In order to prove $L \in \text{BPP}$, it suffices to show that there exists a probabilistic polynomial-time algorithm $A$ and a positive polynomial $p$ such that for every $k$ larger than some $k_0$, 

4-5
1. for any $x \in L_k$, then $\Pr[A(x) = 1] \geq 1/2 + p(k)^{-1}$;

2. for any $x \in \overline{L}_k$, then $\Pr[A(x) = 0] \geq 1/2 + p(k)^{-1}$.

This is because for all $x$ with $|x| \leq k_0$ (that is, instances of bounded length) we can surely just check directly whether $x$ is in $L$ or not (again, since $x$ has bounded length) by having a table with all the answers for all $x$ with length less than $k_0$, while for the remaining $x$ with $|x| > k_0$ we can use $A$. (Better completeness and soundness can be achieved by using $A'(x)$ that runs $A(x)$ poly$(|x|)$ times and takes majority.)

Since $(P,V)$ is a zero-knowledge proof system for $L$, for any (possibly cheating) probabilistic polynomial-time verifier $V'$ there exists a probabilistic polynomial-time algorithm $S_{V'}$ such that the following holds: for every probabilistic polynomial-time distinguisher $C$ and every positive polynomial $q$, for all sufficiently large $k$, and for all $x \in L_k$,

$$\left| \Pr[C(x,S_{V'}(x)) = 1] - \Pr[C(x,PV'[x]) = 1] \right| < \frac{1}{q(k)},$$

or, equivalently,

$$\Pr[C(x,PV'[x]) = 1] - \frac{1}{q(k)} < \Pr[C(x,S_{V'}(x)) = 1] < \Pr[C(x,PV'[x]) = 1] + \frac{1}{q(k)}.$$ 

(Recall that $PV'[x]$ denotes the transcript of the communication between $P$ and $V'$ on the common input $x$, together with the private coin tosses used by $V'$.) In particular, the above holds also for $C = DL$ and $V' = V$.

Given input $x$ with $|x| = k$ and $k > k_0$, we define $A(x) = DL(x,S_{V'}(x))$, where $S_{V'}$ is the simulator for the verifier $V$. Then, choosing $q(k) = 2k^{c_L}$, if $x \in L_k$ we obtain

$$\Pr[A(x) = 1] = \Pr[DL(x,S_{V'}(x))]$$

$$> \Pr[DL(x,PV'[x]) = 1] - \frac{1}{2k^{c_L}}$$

$$> \frac{1}{2} + \frac{1}{k^{c_L}} - \frac{1}{2k^{c_L}}$$

$$= \frac{1}{2} + \frac{1}{2k^{c_L}},$$

and if $x \in \overline{L}_k$, by letting $B = S_{V'}$, we obtain

$$\Pr[A(x) = 1] = \Pr[DL(x,S_{V'}(x))] < \frac{1}{2} - \frac{1}{k^{c_L}}.$$ 

Choosing $p(k) = 2k^{c_L}$, the above two inequalities show that $A$ satisfies the sufficient conditions for demonstrating that $L$ is in $BPP$. 

4-6