Problem 1

A. Give a definition of a negligible function. What does it mean that the function is NOT negligible?

B. Which of the following functions are negligible? (half credit for stating, half credit for proving):

$$F_1(n) = 2^{-128 \log n}$$
$$F_2(n) = 2^{-\log \log n}$$
$$F_3(n) = 1/(\log \log n)$$

1.1 Solution

- Negligible: $\forall c \exists N \forall n > N : f(n) < 1/n^c$ For any polynomial, is smaller than inverse of the polynomial for large enough n.

  Non-negligble: $\exists c \forall N \exists n > N : f(n) \geq 1/n^c$ (informal) Bigger than the inverse of some polynomial for arbitrarily large n. (Not necessarily for all n large enough)

- F_1: Not negl: $= (2^{\log n})^{-128} = 1/n^{128}$ for all n

  F_2: Not negl: $1/\log n > 1/n$ for $n > 1$

  F_3: Not negl: $1/(\log \log n) > 1/n$ for $n > 2$

Problem 2

Let h be a collision-resistant hash function that halves the length of the input. Prove that h is a one-way function.

2.1 Solution

Proof by contrapositive. Suppose h is a hash function that halves the length of the input ($2n$ bits to n bits), but is not a one-way function. We will show that h is not collision resistant. By definition, we are assuming that there exists PPT adversary A such that $Pr[A(h(x)) = x' | h(x') = h(x)] = \epsilon$ for some non-negligible ϵ. We must define a PPT adversary A' which, given h, finds u, v such that $h(u) = h(v)$ and $u \neq v$. The existence of such an adversary, by definition, proves that h is not collision resistant. Define A' as follows:

- Given h, randomly choose $u \in \{0,1\}^{2n}$ and compute $y = h(u)$.
- Compute $A(y) = v$
- Return u, v

Note, A' is in PPT since it's only operations are calling A (which is PPT), one hash evaluation (hash is poly-time computable by definition) and randomly sampling u (a poly-time operation).

We claim that u, v is a collision with at least probability $\epsilon/2 * (1 - 1/2^n)$.

Proof of claim: by our assumption, $h(u) = h(v)$ with probability ϵ, since this is the probability that $A(y)$ returns v satisfying $h(v) = y = h(u)$. What's left is to prove that $u \neq v$ with high probability. Since h maps $2n$ bits to n bits, each output string has 2^n inputs mapping to that output (on average). There are two possibilities: either u is the only element in the pre-image of $h(u)$, or the pre-image contains at least one other element. If the pre-image of $h(u)$ contains at least one other element, there is at least a $1/2$ chance that $A(y)$ returns $v \neq u$ (since A only sees $h(u)$, it is equally likely to return all elements in the pre-image). If u is the only element in the pre-image, $u = v$ always. But, since there are only 2^n possible outputs, there can only be $2^n - 1$ such inputs satisfying this property. Thus the probability that u is the only element in the pre-image of $h(u)$ is at most $(2^n - 1)/2^{2n} < (1/2^n)$, which is negligible. Thus, with overwhelming probability, the pre-image of $h(u)$ contains at least two different elements. In this case, the probability that $u \neq v$ is at least $\epsilon/2$. This proves our claim, and establishes that A' is a PPT machine which finds a collision in h with significant probability.

Problem 3

A. Define a one-time signature scheme.

B. Show that a one-time signature scheme (even restricted to 1-bit document) implies a one-way function.

3.1 Solution

A. A signature scheme is a triple of algorithms (Keygen, Sign, Verify) satisfying:

- Correctness: If $Keygen(1^k, R) = (sk, pk)$ and $Sign(D, sk, R') = \sigma$, then $Verify(\sigma, D, pk) = Accept$ with probability 1.
- Unforgeability: For any PPT adversary A, the probability that A wins the following game is negligible in k:
  - Challenger C chooses random R, computes $Keygen(1^k, R) = (sk, pk)$, and sends pk to A.
  - For $i = 1$ to $l = poly(k)$, A and C do the following:
    - A chooses document D_i and sends D_i to C
    - C samples random R_i and sends $\sigma_i = Sign(D_i, sk, R_i)$ to A
  - A sends D^*, σ^* to C and wins if $D^* \neq D_i$ for all i and $Verify(\sigma^*, D^*, pk) = Accept$

The scheme is "one-time" if security holds only for $l = 1$.

B. Consider $Keygen(1^k, R) = (pk, sk)$. We claim that $f_k(R) = pk$ is a one-way function (ignoring sk). To see this, suppose we have an adversary A which, given pk, can (with non-negligible probability) compute R' such that $Keygen(1^k, R') = (pk, sk')$ for some sk'. Then we can define an adversary A' who, given pk, computes $A(pk) = R'$, then computes $Keygen(1^k, R')$. With non-negligible probability, this gives (pk, sk') such that sk' is a valid secret key for pk. Using this sk', A' can choose any document D^*, compute $Sign(D^*, sk', R^*) = \sigma^*$, and send D^*, σ^* to the challenger. Since (pk, sk') is a valid key pair, correctness of the signature scheme guarantees us that $Verify(\sigma^*, D^*, pk) = Accept$. Thus A' breaks the security of the underlying signature scheme. This proves by contrapositive that if the signature scheme is secure, then f_k is a one-way function.

Problem 4

A unique signature scheme is the one where for each document there is only one signature that is accepted by the verification algorithm. Given a unique signature scheme (where even for maliciously chosen public key, the signature is unique) show how to construct a commitment algorithm. Prove both binding and hiding property.

4.1 Solution

To commit to a bit b, the committer computes $Keygen(1^k, R) = (pk, sk)$, chooses random D and R', computes $\sigma(D) = Sign(D, sk, R')$. They also choose a random p such that $|p| = |\sigma(D)| = n$. If $\langle \sigma(D), p \rangle = b$, they set $p^* = p$. Otherwise, they set $p^* = \bar{p_1} p_2 \dots p_n$. As the commitment, they send D, p^*, pk to the receiver. To decommit, they send $\sigma(D)$. The receiver accepts if $Verify(\sigma(D), D, pk) = Accept$ and they extract the bit b via $\langle \sigma(D), p^* \rangle$. Suppose this scheme were not binding. That is, C sends a σ^* such that $\langle \sigma^*, p^* \rangle = \bar{b}$. If $\sigma^* = \sigma(D)$ then this is a contradiction. Otherwise, if $\sigma^* \neq \sigma(D)$, then $Verify(\sigma(D), D, pk) = Reject$ because $\sigma(D)$ is the unique signature of D under pk. Thus the scheme is binding by contradiction. Suppose the scheme were not hiding. That is, the receiver R can predict $b = \langle \sigma(D), p^* \rangle$ with probability $1/2 + \epsilon$ given D, p^*, pk. Then R can be used as a subroutine to forge signatures under pk. R' simply comnputes $R(D, q, pk)$ and $R(D, q^i, pk)$ for random q, where q^i is q with the i'th bit flipped. This gives $\langle \sigma(D), q \rangle$ and $\langle \sigma(D), q^i \rangle$ with probability at least 2ϵ. Taking the XOR of these two values gives the i'th bit of $\sigma(D)$ with probability 2ϵ. Repeating this for each bit and amplifying gives $\sigma(D)$ with significant probability. R' can then output $D, \sigma(D)$ without ever seeing sk and without any queries.

— Alternate solution with hardcore bits

To commit a bit b: The committer computes $Keygen(1^k, R) = (PK, SK)$. They then pick a random D and compute $\sigma(D) = Sign(D, SK)$. They then pick a random p such that $|p| = |\sigma(D)| = n$. They commit the bit as $\langle \sigma(D), p \rangle \oplus b$. Finally, they output $Com(b, R) = (PK, D, p, \langle \sigma(D), p \rangle \oplus b)$.

To open, the committer reveals $\sigma(D)$. The receiver checks that $Verify(\sigma(D), D, PK) = 1$, then can compute $b = S \oplus \langle \sigma(D), p \rangle =$ themselves to reveal b.

Binding: By unique signatures, there is exactly one possible $\sigma(D)$. Since both D and p are sent in the clear, this means that $\langle \sigma(D), p \rangle$ is deterministic given these values, and hence the commitment can only be opened to b.

Hiding: By security of signatures, $\sigma(D)$ can't be forged even knowing D and PK except with negligible probability. Therefore since p is random they can't predict $\langle \sigma(D), p \rangle$ and break the commitment better than $1/2 + negl$ given D, p and PK.

Problem 5

Explain the 5 round Zero Knowledge Interactive Proof for Graph Isomorphism from lecture. Show correctness, soundness, and Zero Knowledge.

5.1 Solution

Overarching idea: Standard construction needs repetition. Need to do all rounds in parallel. This causes simulation problems. Need to commit to challenges first. In addition, need perfect commitments against the infinite prover. But the actual proving part is the normal iso proof (send H, show how it's iso to G_b for challenge b).

Below for indices i, all are done at once in the same round.

1) P makes $A_0 \sim A_1 \sim G_0$ and sends it
2) V picks random b_i, sends $\Pi_i(A_{b_i})$ for random Π_i to commit to b_i.
3) P picks random c_i, sends random permutation $H_i \sim G_{c_i}$
4) V opens commits to the challenges, sending (b_i, Π_i)
5) P sends $A_0 \sim A_1 \sim G_0$ to prove it's legit, and also sends all answers $H_i \sim G_{b_i}$.

Correctness: P can always do step 5, since $H_i \sim G_0 \sim G_1$, and he builds A correctly.

Soundness: P can't open commits since $A_0 \sim A_1$ and can't tell which V used. So when $c_i \neq b_i$ he fails (half the time, since he doesn't know b and flips a coin for c), and he needs to pass all challenges ($1/2^k$).

Zero Knowledge: Full details in notes, although this outlines the gist of what we were looking for. Two phase simulator, depending on whether V is able to break the commitment or not. If V is honest, Sim goes along until it learns the (committed) challenges, rewinds, builds H_i for those challenges, and continues on with the same challenges, since they were committed. If V is not honest, Sim builds $A_0 \sim G_0$ and $A_1 \sim G_1$ instead. Then Sim does the same learn b_i, rewind to build H_i, forward to learn changed challenges. If any challenge changed, it was opened to both A_0 and A_1, which gives $G_0 \sim A_0 \sim A_i \sim G_1$, and so Sim learns $G_0 \sim G_1$ and can answer any challenge.

Problem 6

- Give a formal definition of a hard-core bit.
- Explain the Goldreich-Levin construction of a hard-core bit.

6.1 Solution

- The hardcore bit $B(x)$ from a one way function $f(x)$ is hard to guess better than random given the output of the OWF. Specifically, for any PPT A, A given $f(x)$ can't guess $B(x)$ more than negligably better than random, i.e. better than $1/2 + negl$. In other words, $\forall A \in PPT, Pr[A(f(x)) = B(x)] = 1/2 + negl < 1/2 + 1/n^c$ for any $c > 0$ and big enough n.
- Construction: Pick a random mask p that is public (shared with adversary A in the game). $B(x, p)$ is constructed as $< x, p >= \sum_i x_i p_i \mod 2$.

  The actual construction in notes had $f'(x, p) = (f(x), p)$ so that $B(x, p)$ is on the same input and p is public. But any variant or mention of p being public or known to the Adversary is fine.

Problem 7

Construct a signature scheme that is secure against 3 rounds of adaptive queries, but is insecure after 4 rounds of adaptive queries. Make sure to define how Keygen, Sign, and Verify work. Recall: Each round of queries can have poly many queries, and can adaptively depending on previous responses received.

7.1 Solution

Assume we have some normal scheme Keygen, Sign, Verify (such as from lecture). We now build Keygen', Sign', Verify' from these with the desired properties.

- $Keygen'(R)$: Use $Keygen(R) = SK, PK$, and also pick 3 random D_1, D_2, D_3 as part of SK' too. So $PK' = PK$ and $SK' = (SK, D_1, D_2, D_3)$
- $Sign'(SK', D) = \begin{cases} Sign(SK, D)||SK & D = D_3 \\ Sign(SK, D)||D_3 & D = D_2 \\ Sign(SK, D)||D_2 & D = D_1 \\ Sign(SK, D)||D_1 & \text{otherwise} \end{cases}$
- $Verify'(PK, D, S||extra)$: Use $Verify(PK, D, S')$, ignoring the second half.

Alternative solution: $Sign'(SK, D) = SK$ if $D = Sign(Sign(Sign(0)))$ else $Sign(SK, D)$. Verify also needs to make sure it works on this edge case, e.g. first check if $Sign(s, D) = s$ to see if we got the SK as the signature, before using normal Verify. However, this solution has a subtle issue, namely that it needs poly many signatures per document for signatures to be polytime (e.g. if the base scheme is unique). For the sake of this problem, we ignored this issue.

- Correctness: Show Verify on any signed document works. Should follow from original Verify, and work on any "trapdoor" cases included.
- Breaks in 4 rounds: Ask for a signature of any D, extract D_1. Then ask for D_1, then D_2, then D_3 similarly. Finally extract SK from the signature of D_3, and you can break anything. (Alternative: Ask for 0, $\sigma(0)$, $\sigma(0)$, etc.)
- Secure in 3 rounds. At a high level skipping a round requires forging a signature, or guessing a D_i, which we can't do with poly many previous queries except with negl.

  You can't guess D_1 except with negligible probability, as it's a random string, or you'd need to forge a signature. Hence you need at least one query to learn D_1 by asking for any random signature. Even with the poly many queries during round 1, none were D_1 except with negl probability (guess/forge), so as above getting D_2 requires guessing, forging, or another query (for signing D_1) in a new round. Same for getting D_3 knowing D_2, and same for getting the signature of D_3 which leaks SK, each requiring another query round or forging/guessing.

  Each forge is negl probability by original privacy, and guessing D_i is negl probability since it's a random string only revealed by the right query. Totals to 8 places with negl chance at winning, and $8 * negl = negl$.
- Alternative solution Secure in 3: Similar idea, negligible chance to forge otherwise at least one query to learn $\sigma(0)$, etc. for each layer. Four places to break, $4 * negl$ is still negl.

Note: Each round can include poly many queries, so making it break after 4 non-adaptive queries fails in one round that asks those queries.

Note: Schemes are not "round aware", i.e. signers don't know any notion of rounds. Even though Naor-Yung implies state, this isn't enough. I.e. it can't tell between 100 signatures all asked in 1 round, or 1 signature at a time in 100 rounds. If you had round awareness, you can literally just have "Act normal for 3 rounds, then also send SK in fourth round". The actual solution(s) follows this gist, but you need to use adaptive queries to mimic "round" knowledge.

Problem 8

An iterated PRG is where a PRG from n bits to $n+1$ bits is applied iteratively, for polynomially many iterations. I.e. $PRG'(s) = PRG(PRG(...(PRG(s))...))$. That is, s is of size n, and the output is of size $s+k$, where k is the number of iterations. Prove that the iterated PRG is a PRG.

8.1 Solution

TYPO: "s + k" should be "n + k". It should still be unambiguous since $|s| = n$ for string s.

For the sake of notation, write them as $G'(s) = G(G(\dots(G(s))\dots)) = G^k(s)$.

We do this by hybrid argument. (Note they hybrids all have to be the same length, so U's are different length.)

$H_0 = G'(s) = G^k(U)$
$H_1 = G^{k-1}(U)$
$\vdots$
$H_{k-1} = G(U)$
$H_k = U$

Note that each "U" is a different length, to ensure that all hybrids are the same length (otherwise distinguishing is trivial). We ignored this.

Now for contradiction suppose G' distinguisher with advantage $\epsilon > negl$. By hybrid, some pair H_i, H_{i+1} gets distinguished by advantage $> \epsilon/k$. Now we want to distinguish G, i.e. distinguish some Z from either G or U. First we guess i, with odds $1/k$. Then we insert Z, getting $G^{k-i-1}(Z)$. If the distinguisher says this is closer to G', that means it's closer to H_i (i.e. k layers of G), and hence Z is closer to G output. Else if it's closer to U it's closer to H_{i+1}, so Z is closer to U. This difference is done with advantage ϵ/k by above. In total this G distinguisher works with $\epsilon/k^2 > negl$ advantage, contradicting G being a PRG.

Alternative solution: Keep each hybrid having a different number of layers. Instead of changing the seed length, instead "outer" layers are removed, and to keep length the same random bits are appended. Then between a pair of hybrids there is a bit that is either from PRG output or uniform random, and the next bit test can be used there. Specifically, given a G' distinguisher, you guess a hybrid pair, insert a bit that's either last bit of G or random, and break last bit test of G using the G' distinguisher, contradiction.

8.1.1 Common issues

Induction: You can't use normal induction, since this isn't true equality but indistinguishably, which includes the "negl". Normal induction lets you repeat any number of times, including exponential, but exponentially many "negl" advantages may no longer be negligible, and hence using just induction is insufficient.

"Fixed guess point": E.g. noting that this is $G(pseudorandom)$, or that the innermost layer is $G(s)$, and breaking there. This does not work, as you don't know where a theoretical G' distinguisher is able to break. E.g. maybe it only can tell the difference better than negl between 3 and 4 layers of G, but is still negl anywhere else. So assuming it can tell the difference between say $k-1$ and k is not necessarily true.