CS 171: Problem Set 2

Due Date: February 8th, 2024 at 8:59pm via Gradescope

1. Negligible/Non-negligible functions

Let $f, g : \mathbb{N} \to \mathbb{R}$ be negligible functions, let $p : \mathbb{N} \to \mathbb{R}$ be a polynomial such that $p(n) > 0$ for all $n \in \mathbb{N}$.

(a) Define $h : \mathbb{N} \to \mathbb{R}$ as $h(n) = f(n) + g(n)$. Prove that h is a negligible function.

(b) Define $h : \mathbb{N} \to \mathbb{R}$ as $h(n) = f(n) \cdot p(n)$. Prove that h is a negligible function.

For each function below, either prove that it is negligible or prove that it is non-negligible (all logarithms are base 2).

(c) $f(n) = n^{-100} + 2^{-n}$

(d) $f(n) = 1.01^{-n}$

(e) $f(n) = 2^{-(\log n)^2}$

(f) $f(n) = e^{-\log^3 n} + e^{-\log^2 n} + e^{-\log n}$

Solution

(a) For a given polynomial P, we want to show that there exists N_P such that for all $n > N_P$, $h(n) < \frac{1}{P(n)}$. As f and g are negligible, then we can find N_f and N_g such that $f(n) < \frac{1}{2P(n)}$ and $g(n) < \frac{1}{2P(n)}$. Thus choose $N_P = \max(N_f, N_g)$, so that for all $n > N_P$, $h(n) = f(n) + g(n) < \frac{1}{P(n)}$. Thus h is negligible.

(b) For a given polynomial P, we want to show that there exists N_P such that for all $n > N_P$, $h(n) < \frac{1}{P(n)}$. As f is negligible, then we can find N_f such that for all $n > N_f$, $f(n) < \frac{1}{p(n)P(n)} \implies f(n) \cdot p(n) < \frac{1}{P(n)}$. Thus choose $N_P = N_f$, so that for all $n > N_P$, $h(n) = f(n) \cdot p(n) < \frac{1}{P(n)}$. Thus h is negligible.

(c) It is non-negligible. Consider $f(n) = n^{-100} + 2^{-n}$.

Claim 0.1 *There exists $N_0 \in \mathbb{N}$ such that for all $n > N_0$, $2^n > n^{100}$.*

Proof Set $N_0 = 100$ and the claim follows. ■

Now, for all $n > N_0$, it follows that

$$\begin{aligned} f(n) &= n^{-100} + 2^{-n} \\ &> n^{-100} + n^{-100} \\ &= 1/2(n^{100}) \\ &:= 1/p(n) \end{aligned}$$

Thus, there exists a polynomial $p(n) = 2n^{100}$ such that for every $N \in \mathbb{N}$, there exists $n > max(N, N_0)$ such that $f(n) > 1/p(n)$. Hence, $f(n)$ is non-negligible.

(d) It is negligible. Consider any polynomial $p(\cdot) : \mathbb{N} \to \mathbb{N}$.
Let $p(n) = a_d n^d + a_{d-1}n^{d-1} + \ldots + a_0$.

Claim 0.2 *There exists $N_0 \in \mathbb{N}$ such that for all $n \geq N_0$, $p(n) < n^{d+2}$.*

Claim 0.3 *There exist N_1 such that for all $n \geq N_1$, $1.01^n > n^{d+2}$*

Proof Observe that for any $n \in \mathbb{N}$ and $n > 100$, $n > \log^2 n$. Set $N_1 = \max(2^{\frac{d+2}{\log(1.01)}}, 100)$. Then, for any $n \geq N_1$, $\log n \geq (d+2)/\log(1.01)$. Thus, $(d+2)\log n \leq \log(1.01)\log^2 n < n\log(1.01)$. ■

It follows from the above two claims that for $n > max(N_0, N_1)$ that $f(n) < 1/p(n)$.

(e) It is negligible.

Claim 0.4 *There exist N_1 such that for all $n \geq N_1$, $2^{\log^2 n} > n^{d+2}$.*

Proof Set $N_1 = 2^{d+2}$. Then, for any $n \geq N_1$, $\log^2 n = (\log n)(\log n) > (d+2)\log n$. ■

It follows from Claim 0.2 and Claim 0.4 that for all $N > \max(N_0, N_1)$, $f(n) < 1/p(n)$.

(f) It is non-negligible. Observe that $f(n) > e^{-\log n}$ and we now show that $e^{-\log n}$ is non-negligible. Notice that $e^{\log n} = n^{\log e} < n^2$ for all $n > 2$. Thus, setting $p(n) = n^2$, we get a proof that it is non-negligible.

■

2. 2-time security?

An encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ over message space $\mathcal{M}$ and ciphertext space $\mathcal{C}$ is said to be 2-time perfectly secure if for any $(m_1, m_2) \in \mathcal{M} \times \mathcal{M}$ and $(m_1', m_2') \in \mathcal{M} \times \mathcal{M}$ such that $m_1 \neq m_2$ and $m_1' \neq m_2'$ and for any $c_1, c_2 \in \mathcal{C}$ the following holds:

$$\Pr[\mathsf{Enc}(K, m_1) = c_1 \wedge \mathsf{Enc}(K, m_2) = c_2] = \Pr[\mathsf{Enc}(K, m_1') = c_1 \wedge \mathsf{Enc}(K, m_2') = c_2].$$

Note that in the above definition the key K is the same for encrypting m_1, m_2 (resp. m_1', m_2'). Consider the following encryption scheme for the message space $\mathbb{Z}_{23}$.

- Gen: Sample two elements $a \xleftarrow{\$} \mathbb{Z}_{23}$ and $b \xleftarrow{\$} \mathbb{Z}_{23}$.
- $\mathsf{Enc}((a, b), m)$: Output $c = a \cdot m + b \mod 23$.
- $\mathsf{Dec}((a, b), c)$: Compute $m = (c - b) \cdot a^{-1} \mod 23$ if a is invertible. Otherwise, output error.

Show the following.

1. Prove that for any message $m \in \mathbb{Z}_{23}$,

$$\Pr[\mathsf{Dec}(K, \mathsf{Enc}(K, m)) = m] = \frac{22}{23}$$

2. Prove that this is 2-time secure.

Solution

1. As long as $a \neq 0$, the scheme recovers the message and the probability of this event happening is $1/23$.

2. Fix any set of messages $(m_0, m_1, m'_0, m'_1) \in \mathbb{Z}_{23}$ such that $m_0 \neq m_1$ and $m'_0 \neq m'_1$ and ciphertexts $c_1, c_2 \in Z_{23}$. We will show that

$$\Pr[\mathsf{Enc}(K, m_1) = c_1 \wedge \mathsf{Enc}(K, m_2) = c_2] = \Pr[\mathsf{Enc}(K, m'_1) = c_1 \wedge \mathsf{Enc}(K, m'_2) = c_2]$$

Let (A, B) be the uniform random variables over $\mathbb{Z}_{23}$. The random variable denoting the key K is given by (A, B).

$$\begin{aligned}\Pr[\mathsf{Enc}(K, m_1) = c_1 \wedge \mathsf{Enc}(K, m_2) = c_2] &= \Pr_K[\mathsf{Enc}(K, m_1) = c_1 \wedge \mathsf{Enc}(K, m_2) = c_2] \\ &= \Pr_{A,B}[Am_1 + B = c_1 \wedge Am_2 + B = c_2] \\ &= \frac{1}{23^2}\end{aligned}$$

By a similar argument, we can show that $\Pr[\mathsf{Enc}(K, m'_1) = c_1 \wedge \mathsf{Enc}(K, m'_2) = c_2] = \frac{1}{23^2}$.

■

3. Getting Adversarial

Alice the Frog is very excited to share her new encryption scheme with you. You are responsible for convincing her it is insecure. The objective of this question is to familiarize you with the security framework of computational indistinguishibility. Download the `zip` file at `eecs171.com/assets/homework/hw2.zip`. Fill in the `TODO`s and upload your completed `scheme2.py` and `adversary.py` to Gradescope. You are provided with 5 files:

- `scheme1.py` specifies Alice's encryption scheme. Do not change this code.
- `scheme2.py` is where you should write the decryption scheme corresponding to the encryption scheme in `scheme1.py`.
- `correctness.py` provides you with a basic sanity test to confirm that the decryption scheme you wrote successfully recovers a plaintext encrypted with Alice's encryption scheme. Typically, we require that correctness is enforced for every message, but here, we are only checking for random messages.
- `security.py` contains the computational indistinguishibility security game which invokes the adversary. Typically, we require that the adversary succeeds with probability only non-negligibly better than 1/2, but here, we are checking that the adversary succeeds with probability 1.

- `adversary.py` is where you will write the adversarial code which is invoked by `security.py` in the security game. As shown in the skeleton code, note that the adversary is called twice.

Note: When making your submission, highlight both files and compress them directly, rather than zipping a folder containing the files.

Solution

`scheme2.py`

```
def dec(key, ciphertext):
"""Decrypt the message using the given encryption scheme."""
aes_ciphertext, xor_all_bytes = ciphertext

# Initialize AES cipher with key and a zero IV
cipher = Cipher(algorithms.AES(key), modes.ECB(), backend=default_backend())
encryptor = cipher.encryptor()

# Encrypt the zero value
zero_value_encrypted = encryptor.update(b'\x00' * 16) + encryptor.finalize()

# XOR the encrypted message with the encrypted zero value
decrypted_message = bytes([e ^ z for e, z in zip(aes_ciphertext, zero_value_e

# Verify the XOR of all bytes
xor_check = 0
for byte in decrypted_message:
    xor_check ^= byte

if xor_check != xor_all_bytes:
    raise ValueError("Decryption failed: XOR check mismatch")

return decrypted_message
```

`adversary.py`

```
def adversary(call, encrypted_message=None):
"""Adversary function."""
m0 = b'\x00'*7
m1 = b'\x01'*7
if call == 1:
    # First call: Adversary specifies two messages
    return m0, m1
elif call == 2:
    # Second call: Adversary guesses which message was encrypted
    # Extract the XOR part from the encrypted message
    _, xor_part = encrypted_message

    # Compute the XOR of all bytes for both messages received in the first c
```

```
def xor_bytes(message):
    xor_result = 0
    for byte in message:
        xor_result ^= byte
    return xor_result
xor_m0 = xor_bytes(m0)
xor_m1 = xor_bytes(m1)

# The message whose XOR matches the XOR part of the encrypted message is
if xor_m0 == xor_part:
    return 0
elif xor_m1 == xor_part:
    return 1
```

Note that this is just a solution that works and there are different, simpler, more complicated, etc. solutions that also work! ■