Problem 1: Pseudorandom Fun(ctions)

Given a pseudorandom function $F_s : \{0,1\}^{k+\lceil \log k \rceil} \mapsto \{0,1\}$, construct a pseudorandom function $F'_s : \{0,1\}^k \mapsto \{0,1\}^k$ and prove that it is secure.

Problem 2: GGM and Prefix-Constrained PRFs

A PRF $F : \{0,1\}^k \times \{0,1\}^k \mapsto \{0,1\}^k$ is said to be a prefix-constrained PRF if given the PRF key it is possible to generate a constrained PRF key $K_\pi$ which lets you evaluate the PRF only at inputs which have a specific prefix $\pi$. More precisely, a prefix-constrained PRF has the following algorithms:

- **Setup** : $\text{Setup}(1^k)$ outputs a key $K \leftarrow \{0,1\}^k$
- **Constrain** : For any string $\pi$ such that $|\pi| \leq k$, $\text{Constrain}(K, \pi)$ outputs a key $K_\pi$
- **Evaluate** : $\text{Eval}(K_\pi, x)$ outputs $F(K, x)$ iff. $x = \pi \circ t$ for some $t \in \{0,1\}^{k-|\pi|}$, else output fail

The security notion for a constrained PRF key $K_\pi$ is that it should reveal no information about the PRF evaluation at points that do not have the prefix $\pi$. More precisely, for any string $\pi$ such that $|\pi| \leq k$, let $X_\pi$ be the set of all $x \in \{0,1\}^k$ that do not have $\pi$ as their prefix. We say $F : \{0,1\}^k \times \{0,1\}^k \mapsto \{0,1\}^k$ is a spring-break-secure prefix-constrained PRF if for all PPT $\mathcal{A}$, there exists a negligible $\nu()$ such that

$$|\Pr[\mathcal{A}(1^k) \text{ is in Exp 1} : b' = 0] - \Pr[\mathcal{A}(1^k) \text{ is in Exp 2} : b' = 0]| = \nu(k)$$

where

**Exp 1**

1. Choose key $K \leftarrow \{0,1\}^k$
2. $\mathcal{A}(1^k)$ chooses a prefix $\pi$ with $|\pi| \leq k$ and obtains $K_\pi = \text{Constrain}(K, \pi)$
3. $\mathcal{A}(1^k)$ adaptively queries $F(K, \cdot)$ on any inputs $x_1, \ldots, x_q \in X_\pi$ and obtains values $F(K,x_i)$ for $1 \leq i \leq q$
4. $\mathcal{A}$ outputs a guess $b'$

**Exp 2**

1. Pick a random function $R : \{0,1\}^k \mapsto \{0,1\}^k$
2. $\mathcal{A}(1^k)$ chooses a prefix $\pi$ with $|\pi| \leq k$ and obtains $K_\pi = \text{Constrain}(K, \pi)$
3. $\mathcal{A}(1^k)$ adaptively queries $R(\cdot)$ on any inputs $x_1, \ldots, x_q \in X_\pi$ and obtains values $R(x_i)$ for $1 \leq i \leq q$
4. $\mathcal{A}$ outputs a guess $b'$

In this problem, we will prove that the GGM PRF we have seen in class is also a prefix-constrained PRF. Recall that the GGM PRF is obtained as follows: Start with a length-doubling PRG $G : \{0,1\}^k \mapsto \{0,1\}^{2^k}$ and for any $s \in \{0,1\}^k$ outputs a string of length $2k$; We will call
the first half as $G_0(s)$ and second half as $G_1(s)$. Let input be $x = x_1x_2\ldots x_k$ where each $x_i \in \{0, 1\}$, then the PRF, with key $K$ is defined as follows:

$$F(K, x_1x_2\ldots x_k) = G_{x_k}(\ldots G_{x_2}(G_{x_1}(K))\ldots)$$

a. For the GGM PRF, what could be the constrained key $K_0$ that lets you evaluate $F(K, x)$ for all $x$ starting with a 0? How will you evaluate the PRF with this constrained key?

b. Design the Constrain($K, \pi$) algorithm for any prefix $\pi$ with $|\pi| \leq k$ for the GGM PRF.

c. Describe the corresponding Eval($K_\pi, x$) algorithm.

d. Prove that your prefix-constrained PRF is spring-break-secure.

### Problem 3: PRFs/PRPs as MACs

A message authentication code (MAC) is a method for ensuring that the data received over the network came from the right person. More precisely, it is an object that satisfies the following properties: Suppose Alice and Bob share a secret $s$, and Bob wants to send message $m$ to Alice. Then $MAC_s(m)$ is an efficiently computable function such that even if an active attacker Eve queries it on a set of messages of its choice, it still cannot authenticate any message not explicitly queried.

To give a more formal definition, let us first extend our notation. If $A \cdot (\cdot)$ is an oracle Turing machine, then by $Q \leftarrow A^n(x)$ we denote the contents of $A$’s query tape upon termination with oracle $O$ and input $x$. Now: A function family $\{M_s(\cdot)\}$ is a MAC if for all PPT $A$, there exists a negligible function $\nu(k)$ such that

$$\Pr[s \leftarrow \{0, 1\}^k; \{(m, x), Q\} \leftarrow A^k^n : x = M_s(m) \text{ and } (m, x) \notin Q] \leq \nu(k)$$

a. Let $\{F_s : \{0, 1\}^{|s|} \rightarrow \{0, 1\}^{|s|}\}$ be a PRF family. Show that it is a MAC.

b. Is it the case that a MAC is a PRF? Prove your answer.

c. Is it true that if MACs exist then PRFs exist? Briefly defend your answer.

### Problem 4: Collision-resistant hash functions

Consider the following hash function family for hashing integers:

- $Gen(1^k)$: generate 2 $k$-bit primes $p, q$. Let $n = pq$. Choose random $y \leftarrow \mathbb{QR}_n$ and output $n, y$.
- $H_{n, y}(x) = y^x \mod n$

a. Recall that the RSA assumption says that, given an RSA public key $(n, e) \leftarrow G_{RSA}(1^k)$ 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 probabilistic polynomial-time adversaries $A$, there exists a negligible function $\nu(\cdot)$ such that

$$\Pr[(n, e) \leftarrow G_{RSA}(1^k); y \leftarrow \mathbb{Z}_n^*; x \leftarrow A(n, e, y) : y = x^e] = \nu(k).$$

Prove that if the RSA assumption holds, then the hash function described above is collision-resistant (as usual, use a reduction).
Problem 5: Merkle Trees

Let \((G, H(\cdot))\) be a family of collision-resistant hash functions where \(H_{pk} : \{0, 1\}^* \rightarrow \{0, 1\}^k\) for \(pk \in G(1^k)\).

For \(n = 2^\ell\), the Merkle hash \(\text{Merkle}_{pk,n}(x_0, \ldots, x_{n-1})\) defined by \((G, H(\cdot))\) is an algorithm that takes as input a public key \(pk\) of the hash function \(H_{pk}\) and \(n\) binary strings \(x_0, \ldots, x_{n-1}\). For every binary string \(s\) of length at most \(\ell\), compute its label \(u_s\) as follows: if \(|s| = \ell\), then \(u_s = H_{pk}(x_s)\); otherwise \(u_s = H_{pk}(u_{s0} \circ u_{s1})\). Output \(u_\varepsilon\) (recall that \(\varepsilon\) is the empty string).

The authenticating path of \(x_i\) in \(\text{Merkle}_{pk,n}(x_0, \ldots, x_{n-1})\) is the set of values \((u_{t_1}, \ldots, u_{t_\ell})\) where each \(t_j\) is a string of length \(j\) where the first \(j-1\) bits agree with the first \(j-1\) bits of \(i\), and the last bit is different (so for example if \(i = 00111\), then \(t_1 = 1, t_2 = 01, t_3 = 000, t_4 = 0010, t_5 = 00110\)). These are just the labels of the nodes whose siblings are on the path from the \(i\)th leaf to the root.

To verify an authenticating path, that is, to verify that a given string \(x\) is the \(i\)th string in the set of strings that was hashed together to obtain the value \(v\), \(\text{MVerify}_{pk,n}(v, x, i, a, a')\) proceeds as follows: first compute \(u_i = H_{pk}(x)\). For \(j\) from \(\ell - 1\) to 0: Let \(I_j\) be the \(j\)-bit prefix of the binary string \(i\). Compute \(u_I = H_{pk}(u_{I0} \circ u_{I1})\) (if the \(j+1\)st bit of \(i\) is 0, then \(u_{I0} = u_{Ij+1}\) computed in the previous iteration, while \(u_{I1}\) was given as part of the authenticating path; if the \(j+1\)st bit of \(i\) is 1, then it’s the other way around). Finally, accept if \(u_{\varepsilon} = v\), and reject otherwise. Intuitively, this procedure iteratively computes the label on each node on the path from the \(i\)th leaf to the root from the labels of its children, and accepts if the label it computed for the root is \(v\).

a. What are the domain and range of \(\text{Merkle}_{pk,n}\)? Show that \((G, \text{Merkle}(\cdot), n)\) is a family of collision-resistant hash functions for these domain and range, assuming that \((G, H(\cdot))\) is a collision-resistant hash function family.

b. Prove that, assuming that \((G, H(\cdot))\) is a collision-resistant hash function family, \(\text{MVerify}_{pk,n}\) is sound. Namely, no \(A\) can produce an authenticating path for the same Merkle root \(v\) but conflicting \(i\)th documents \(x\) and \(x'\). More formally, show that, for every probabilistic polynomial time \(A\) there exists a negligible \(\nu\) such that

\[
\Pr[pk \leftarrow G(1^k); (v, x, x', i, a, a') \leftarrow A(pk) : x \neq x' \wedge \text{MVerify}_{pk,n}(v, x, i, a) \wedge \text{MVerify}_{pk,n}(v, x', i, a') = \nu(k)]
\]