Instructions

- You are allowed to work in groups of size at most 2.
- The assignments must be typed in Latex, and the resulting pdf must be submitted on Gradescope.
- The bonus questions are somewhat challenging, and you are recommended to attempt them only after solving all the other problems.
- **Plagiarism policy**: You should not discuss your solutions with other group members. Sharing your solutions with other group members is strictly not allowed, and if there are significant similarities in two or more submissions, all relevant group members will be penalized.

  You can refer to resources online, but you should write your solutions in your own words (and also cite the resources used).
- For the last question, you can either attempt the theoretical question (Part 1, in which case you must submit the proofs in the pdf), or the coding question (Part 2, in which case you must submit a brief description of your solution in the pdf, and submit your code separately). Code submission instructions, as well as the executable, will be provided by 16th October.

Notations

Given a finite set S, let $x \leftarrow S$ denote a uniformly random sample x drawn from S. Let p be a prime. The set $\{0, 1, \ldots, p-1\}$ is represented by $\mathbb{Z}_p$, while $\mathbb{Z}_p^* = \{1, 2, \ldots, p-1\}$. For an element $g \in \mathbb{Z}_p^*$, let $\langle g \rangle_p = \{g^0, g^1, g^2, \ldots\} \subseteq \mathbb{Z}_p^*$.

Questions

1. (10 marks) **ECBC-MAC: Proof of security**

   In the previous assignment, we had stated that the encrypted CBC-MAC is a secure MAC scheme for signing unbounded length messages. In this assignment, we will prove its security. Let us first recall the ECBC-MAC scheme. It uses a PRF $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$, supports message space $\mathcal{M} = (\{0,1\}^n)^*$, and has key space as $\{0,1\}^n \times \{0,1\}^n$.

   Encrypted CBC-MAC: MAC scheme for message space $(\{0,1\}^n)^*$

   - $\mathsf{Sign}(m = (m_1, \ldots, m_\ell), k = (k_1, k_2))$: Let $y_1 = F(m_1, k_1)$. For each $i \in [2, \ell]$, compute $y_i = F(m_i \oplus y_{i-1}, k_1)$. Finally, output $\sigma = F(y_\ell, k_2)$.
   - $\mathsf{Verify}(m = (m_1, \ldots, m_\ell), \sigma, k = (k_1, k_2))$: Let $y_1 = F(m_1, k_1)$. For each $i \in [2, \ell]$, compute $y_i = F(m_i \oplus y_{i-1}, k_1)$. Output 1 iff $\sigma = F(y_\ell, k_2)$.

   We can view the ECBC-MAC scheme as an instantiation of the 'hash then sign' paradigm. Given a message $m = (m_1, \ldots, m_\ell)$, the string y_ℓ obtained in the construction above is a 'hash' of the message, and $F(y_\ell, k_2)$ is a signature on y_ℓ. Therefore, in order to prove that ECBC-MAC is strongly unforgeable, it suffices to show that the keyed computation mapping m to y_ℓ is a secure UHF (and then, we can use Theorem 22.01 from Lecture 22).

   Let $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ be a secure pseudorandom function. Show that the following function family is a secure UHF, assuming F is a secure PRF. The key space is $\{0,1\}^n$, the input space

is $(\{0,1\}^n)^*$, output space is $\{0,1\}^n$. The evaluation on input $x = (x_1, x_2, \ldots, x_\ell)$ is as follows: set $y_1 = F(x_1, k)$, and for all $i > 1$, $y_i = F(x_i \oplus y_{i-1}, k)$. Output y_ℓ as the UHF evaluation.

2. (15 marks) **Hashing: a security notion in-between UHFs and CRHFs**

In class, when we discussed UHFs and CRHFs, one of the students proposed the following security definition for security of hash functions. Let $\mathcal{H} = \{H_k : \{0,1\}^\ell \to \{0,1\}^n\}_{k \in \mathcal{K}}$ be a family of keyed compressing functions (that is, $\ell > n$). We say that this function family satisfies U/CR-HF security if no p.p.t. adversary $\mathcal{A}$ can win the following security game with non-negligible probability:

U/CR-HF

- **(Setup)** Challenger chooses a key $k \leftarrow \mathcal{K}$.
- **(Hash function queries)** Adversary is allowed polynomially many queries. For every query $x_i \in \{0,1\}^\ell$, the challenger sends $H_k(x_i)$.
- **(Collision)** After polynomially many queries, adversary sends two distinct strings x_0^*, x_1^*, and wins if $H_k(x_0^*) = H_k(x_1^*)$.

Figure 1: Intermediate security notion for hash functions

1. (4 marks) Construct a universal hash function family that does not satisfy the U/CR-HF security notion. Provide the construction formally (no need to prove that it is a UHF), and show that there exists a p.p.t. adversary that wins the U/CR-HF security game with non-negligible probability.
2. (4 marks) Let $\mathcal{H} = \{H_k : \{0,1\}^\ell \to \{0,1\}^n\}_{k \in \mathcal{K}}$ be a secure collision resistant hash function family with key space $\mathcal{K}$, and $\ell > n$. Construct a new hash function family $\mathcal{H}' = \{H'_k : \{0,1\}^\ell \to \{0,1\}^n\}_{k \in \mathcal{K}'}$ with different key space $\mathcal{K}'$ such that $\mathcal{H}'$ satisfies U/CR-HF security, but is not a secure CRHF. Again, just describe the function family formally (no need to prove U/CR-HF security), and discuss why it is not a secure CRHF.
3. (7 marks) Let $\mathcal{H} = \{H_k : \{0,1\}^{2n} \to \{0,1\}^n\}_{k \in \mathcal{K}}$ be a secure U/CR-HF family with key space $\mathcal{K}$. Show that the Merkle-Damgard transformation can be used to construct a secure U/CR-HF family with key space $\mathcal{K}$, input space $(\{0,1\}^n)^{\geq 2}$ and output space $\{0,1\}^n$.

3. (15 marks) **Collision Resistant Hashing from number-theoretic assumptions**

Let $\mathsf{Gen\text{-}Safe\text{-}Prime}(1^n)$ be a p.p.t. algorithm that takes as input 1^n, and generates an n-bit prime p such that $p = 2q + 1$ and q is also prime.[1] Consider the following number-theoretic assumption (known as the Discrete Log problem over prime-order subgroups of $\mathbb{Z}_p^*$):

For any p.p.t. adversary $\mathcal{A}$, there exists a negligible function $\mu(\cdot)$ such that for all n,

$$\Pr\left[\mathcal{A}(p, g, g^a) = a \;:\; \begin{array}{c} p \leftarrow \mathsf{Gen\text{-}Safe\text{-}Prime}(1^n) \\ g \leftarrow \mathbb{Z}_p^* \text{ s.t. } |\langle g \rangle_p| = q \\ a \leftarrow \mathbb{Z}_q \end{array}\right] \leq \mu(n)$$

Below, we define a hash function family. The function family is parameterized by a safe prime $p \leftarrow \mathsf{Gen\text{-}Safe\text{-}Prime}(1^n)$ and $g \in \mathbb{Z}_p^*$ such that $|\langle g \rangle_p| = q$. The input space is $\mathbb{Z}_q^n$, output space is $\mathbb{Z}_p^*$. Each key consists of n uniformly random elements $(x_1, x_2, \ldots, x_n)$ where $x_i \leftarrow \langle g \rangle_p$ for each $i \in [n]$. The hash function evaluation using key $k = (x_1, x_2, \ldots, x_n)$, on input $(\alpha_1, \alpha_2, \ldots, \alpha_n)$ is $(\prod_i x_i^{\alpha_i}) \mod p$.

Show that if there exists a p.p.t. algorithm $\mathcal{A}$ that breaks the collision-resistance property of this hash function family with probability ϵ, then there exists a p.p.t. algorithm $\mathcal{B}$ that breaks the discrete log assumption with probability $\epsilon - \mathsf{negl}(n)$.

[1] Strictly speaking, it is not known if there are infinitely many 'safe primes', and therefore we don't know such an algorithm for large n. We know safe primes for sufficiently large n, which suffices for practical purposes. Also, there are other 'groups' that can be used instead.

Easier Version (8 marks) Show that if there exists a p.p.t. algorithm $\mathcal{A}$ that breaks the collision-resistance property of this hash function family with non-negligible probability ϵ, then there exists a p.p.t. algorithm $\mathcal{B}$ that breaks the discrete log assumption with non-negligible probability ϵ'. Unlike the 'full-credit' version, here ϵ' can be equal to $\epsilon/\mathsf{poly}(n)$.

4. (10 marks) Attempt **either** Part 1 **or** Part 2. If both are attempted, then only Part 2 will be graded.

   1. **A CCA-secure 'MAC-then-Encrypt' scheme**

      In class, we saw that 'MAC-then-Encrypt' is, in general, insecure against chosen-ciphertext attacks. However, for specific MAC and encryption schemes, the combination is CCA secure. Let $F_1 : \{0,1\}^n \times \mathcal{K}_1 \to \{0,1\}^n$ and $F_2 : \{0,1\}^n \times \mathcal{K}_2 \to \{0,1\}^{2n}$ be two secure pseudorandom function with key spaces $\mathcal{K}_1$ and $\mathcal{K}_2$ respectively.

      Consider the following encryption scheme with message space $\{0,1\}^n$.

      - $\mathsf{Enc}(m, (k_1, k_2))$: It uses the first key k_1 for the signature computation; that is, it sets $\sigma = F_1(m, k_1)$. Next, it sets $z = \sigma \,||\, m$, chooses $r \leftarrow \{0,1\}^n$ and sets $\mathsf{ct}_0 = r, \mathsf{ct}_1 = (z \oplus F_2(r, k_2))$. The ciphertext is $\mathsf{ct} = (\mathsf{ct}_0, \mathsf{ct}_1)$.
      - $\mathsf{Dec}((\mathsf{ct}_0, \mathsf{ct}_1), (k_1, k_2))$: The decryption algorith first computes $z = \mathsf{ct}_1 \oplus F_2(\mathsf{ct}_0, k_2)$. Let $z = \sigma \,||\, m$. Next, it checks if $\sigma = F_1(K_1, m)$. If so, it outputs m, else it outputs $\perp$.

      Prove that the above scheme is secure against chosen-ciphertext attacks, assuming the F_1 and F_2 are secure pseudorandom functions.

   2. **Coding Question: Padding Oracle Attack**

      Consider the following encryption scheme:

      - $\mathsf{Enc}(m, k)$: The message m is an arbitrary sequence of bytes. Here the key k is a 16 byte string. We use AES in CBC mode to encrypt this message. Since AES can only handle messages whose length (in bits) is a multiple of 128, we have to pad m appropriately.

        Padding Scheme: Instead of padding at the right-most end, we would instead pad at the left-most end (as suggested in class). Let p be the number of bytes to be padded – then include the number p (in binary) in each of the p bytes.

        Examples:

        - m = (11 42 33 01 89 12). This message is 6 bytes long. We need to pad it with 10 bytes. The resulting message m' would be m' = (10 10 10 10 10 10 10 10 10 10 11 42 33 01 89 12).
        - m = (02 02 00 00 00 00 00 00 00 00 00 00 00 00 00 16). This message is 16 bytes long. Add a new padding block to avoid ambiguity. The padded message m' = (16 16 ... 16 16 02 02 00 00 00 00 00 00 00 00 00 00 00 00 16).

      - $\mathsf{Dec}(\mathsf{ct}, k)$: Decryption algorithm simply decrypts the ciphertext to obtain the padded message $m' = (y_1, y_2, ..., y_\ell)$ where each y_i is 16 bytes long. It checks if y_1 is a valid padded string. That is, check that the first byte of y_1 is a number between 1 and 16. If the number is z, then check that the next $z - 1$ bytes after it have the value z. If any of these is violated, output "Error: Bad Padding". Otherwise, output the decrypted string (without the padding).

      You are given two things:

      - a ciphertext, which is an encryption of some message using key k
      - an executable, which implments the decryption function with key k hardwired. The executable, however, does not output the decrypted message. Instead, if there is a bad padding error, it simply outputs the error, otherwise it outputs nothing. It only tells you if the decrypted message had a valid padding or not.

      Use the executable to learn the encrypted message. Give a brief description of your algorithm in the pdf. Additionally, you must also submit your code. The executable, together with instructions for submitting the code, will be provided by 16th October.