Problem 1: Merkle-Damgård and the ROM

In the lecture, I explained the random oracle heuristic which suggests to model a hash function as a random oracle. It should be added that a (preferable) refinement of this heuristic is to model the compression function itself as a random oracle, and to model the hash function as some function constructed based on that compression function (using, e.g., Merkle-Damgård). The reason behind this is that constructions like Merkle-Damgård do not produce functions that behave like random functions (even if the underlying compression function is a random function).

Give an example why a hash function $H$ constructed using the Merkle-Damgård construction should not be modeled as a random oracle. More precisely, find a cryptographic scheme which is secure when $H$ is a random oracle (no security proof needed), but which is insecure when $H$ is a Merkle-Damgård construction (even if the compression function is a random oracle).

**Hint:** Consider the construction of MACs from hash functions that is insecure when the hash function is constructed with Merkle-Damgård.

Problem 2: ElGamal FDH

Bob studied the RSA-FDH construction. He notices that RSA-FDH essentially does the following: To sign a message $m$, it decrypts $H(m)$ using textbook RSA, and to check a signature $\sigma$, it encrypts $\sigma$ and compares the result with $H(m)$.

This lead him to the following idea: Instead of textbook RSA, he uses ElGamal in the construction of FDH, because ElGamal is more secure (it is IND-CPA secure, after all).

Why is the resulting scheme “ElGamal-FDH” bad?

Problem 3: Random oracle model

Write down the definition of IND-CPA security in the random oracle model (for symmetric encryption schemes).

Problem 4: Security proof in the ROM [Bonus problem]

This is a bonus problem.
Fix a hash function $H : \{0,1\}^* \rightarrow \{0,1\}^n$. We define the following block cipher with message and key space $\{0,1\}^n$:

- **Encryption:** To encrypt $m \in \{0,1\}^n$ under key $k$, choose a random $r \in \{0,1\}^n$ and return the ciphertext $c := (r, m \oplus H(k\|r))$.
- **Decryption:** To decrypt $c = (r, c')$ with key $k$, compute and return $m := H(k\|r) \oplus c'$.

Below is a proof that this encryption scheme is IND-CPA secure in the random oracle model. Fill in the gaps. (The length of the gaps is unrelated to the length of the text to be inserted.)

**Proof.** In the first game, we just restate the game from the IND-CPA security definition (in the random oracle model).

**Game 1.** $\diamond$

To show that the encryption scheme is IND-CPA secure, we need to show that

$$|\Pr[b = b' \text{ Game 1}] - \frac{1}{2}|$$

is negligible

(1)

As a first step, we replace the random oracle.

**Game 2.** Like Game 1 except that we define the random oracle $H$ differently: $\diamond$

We have $\Pr[b = b' \text{ Game 1}] = \Pr[b = b' \text{ Game 2}]$.

One can see that the adversary cannot guess the key $k$ (where $k$ is the key used for encryption in Game 2), more precisely, the following happens with negligible probability:

“The adversary invokes $H(x)$ with $x = k\|r'$ for some $r'$.” (We omit the proof of this fact.)

Let $r_0$ denote the value $r$ that is chosen during the execution of $c := E^H(k, m_b)$ in Game 2. Consider the following event: “Besides the query $H(k\|r_0)$ performed by $c := E^H(k, m_b)$, there is another query $H(x)$ with $x = k\|r'_0$ (performed by the adversary or by the oracle $E^H(k, \cdot)$).” This event occurs with negligible probability. Namely, the adversary make such $H(x)$ queries with negligible probability because $\diamond$, and the oracle $E^H(k, m_b)$ makes such $H(x)$ queries with negligible probability because $\diamond$.

Thus, the response of the $H(k\|r_0)$-query performed by $c := E^H(k, m_b)$ is a random value that is used nowhere else (except with negligible probability). Thus, we can replace that value by some fresh random value.

**Game 3.** Like Game 2 except that we replace $c := E^H(k, m_b)$ by $r_0 \leftarrow \{0,1\}^n$, $h^* \leftarrow \{0,1\}^n$, $c \leftarrow (r_0, m_b \oplus h^*)$. $\diamond$

We have that $|\Pr[b = b' \text{ Game 2}] - \Pr[b = b' \text{ Game 3}]|$ is negligible.

To get rid of $m_b$ in Game 3, we use the fact that $h^*$ is chosen uniformly at random and XORed on $m_b$.

That is, we can replace $m_b \oplus h^*$ by $\diamond$.

**Game 4.** Like Game 3 except that we replace $c \leftarrow (r_0, m_b \oplus h^*)$ by $\diamond$.

We have that $\Pr[b = b' \text{ Game 4}] = \Pr[b = b': ??]$. Notice that $b$ is not used in Game 4 thus we have that $\Pr[b = b' \text{ Game 4}] = \diamond$.

Combining the equations we have gathered, (1) follows. $\square$
Problem 5: Tree-based signatures

This problem refers to the tree-based construction of signature schemes from one-time signatures from Construction 7 in the lecture notes. You may assume that Lamport’s signature scheme (Construction 4 in the lecture notes) is used as the underlying one-time signature scheme. (Where all messages are first hashed with a hash function \( H \) before signing with Lamport’s scheme in order to fit in the message space.)

(a) Assume someone has implemented the signature scheme incorrectly as follows: Instead of using randomness from the pseudorandom function \( F \) for the signing and key-generation algorithm, it runs signing and key-generation normally (i.e., as probabilistic algorithms, with fresh randomness each time it is invoked).

Explain how to break the signature scheme. More precisely, show how to sign an arbitrary message \( m \) by performing only signature queries for messages \( m’ \neq m \).

Note: Be explicit: describe all the actions and computations the adversary has to perform. (E.g., give the adversary in pseudocode.) It is not sufficient to say something like: “since two signatures are produced using the same key with a one-time signature scheme, the adversary can break the scheme”. Remember that the underlying scheme is Lamport’s one-time signature scheme.

(b) Bonus problem: Lamport’s signature scheme has public keys consisting of \( 2\eta \) \( \eta \)-bit blocks (assuming that the one-way function \( f \) has domain and range \( \{0,1\}^\eta \)). But it signs only messages consisting of a single \( \eta \)-bit block. In the tree-based construction, we need to sign two Lamport public keys, i.e., \( 4\eta \) \( \eta \)-bit blocks. Normally we solve this by converting Lamport’s scheme into a one-time signature scheme for long messages by hashing the messages to be signed.

Here we explore a different possibility. Instead of hashing the \( 4\eta \times \eta \) bits, we XOR the blocks together. That is, from Lamport’s scheme \((KG_{Lamport}, Sign_{Lamport}, Verify_{Lamport})\) we construct a one-time signature scheme \((KG_1, Sign_1, Verify_1)\) for \( 4\eta \times \eta \)-bit messages as follows:

\[
KG_1 := KG_{Lamport} \quad Sign_1(sk, m_1 \parallel \ldots \parallel m_{4\eta}) := Sign_{Lamport}(sk, \bigoplus_{i=1}^{4\eta} m_i) \quad \text{for} \quad m_1,\ldots,m_{4\eta} \in \{0,1\}^\eta.
\]

\[
Verify_1(pk, m_1 \ldots m_{4\eta}, \sigma) := Verify_{Lamport}(pk, \bigoplus_{i=1}^{4\eta} m_i, \sigma).
\]

Now we can construct the tree-based signature scheme \((KG_{tree,} Sign_{tree}, Verify_{tree})\) from \((KG_1, Sign_1, Verify_1)\) without needing a hash function (as in Construction 7 in the lecture notes).

Your task: Break the resulting \((KG_{tree,} Sign_{tree}, Verify_{tree})\).

Note: It is not sufficient to just show that \((KG_1, Sign_1, Verify_1)\) is insecure. You have to break \((KG_{tree,} Sign_{tree}, Verify_{tree})\). All the other comments from the note of (a) also apply.