Problem 1: 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?

Solution. The first problem is that ElGamal expects as a ciphertext a pair of integers. Thus we have to make sure that \( H(m) \) is actually a pair of integers (in the suitable range), otherwise decryption (signing) will fail due to a malformed ciphertext.

But even if we have found a hash function \( H \) that outputs suitable pairs of integers, verification will always fail. Since ElGamal is randomized, it does not hold that decrypting and then encrypting again yields the original ciphertext. Or to put it differently: For any \( m \), the probability that encrypting \( m \) with ElGamal yields \( H(m) \) is negligible.

Problem 2: Authentication in WEP (bonus problem)

In the WEP-protocol (used for securing Wifi, now mostly replaced by WPA), messages are “encrypted” using the following procedure: First, a key \( k \) is established between the parties \( A \) and \( B \). (We do not care how, for the purpose of this exercise we assume that this is done securely.) Then, to transmit a message \( m \), \( A \) chooses an initialization vector \( IV \) (we do not care how) and sends \( IV \) and \( c := \text{keystream} \oplus (m \parallel \text{CRC}(m)) \). Here \( \text{keystream} \) is the RC4 keystream computed from \( IV \) and \( k \) (we do not care how).

The function \( \text{CRC} \) is a so-called cyclic redundancy check, a checksum added to the WEP protocol to ensure integrity. We only give the important facts about \( \text{CRC} \) and omit a full description. Each bit of \( \text{CRC}(m) \) is the XOR of some of the message bits. Which messages bits are XORed into which bit of \( \text{CRC}(m) \) is publicly known. (In other words, the \( i \)-th bit of \( \text{CRC}(m) \) is \( \bigoplus_{j \in I_i} m_j \) for a publicly known \( I_i \).)

An adversary intercepts the ciphertext \( c \). He wishes to flip certain bits of the message (i.e., he wants to replace \( m \) by \( m \oplus p \) for some fixed \( p \)). This can be done by flipping the corresponding bits of the ciphertext \( c \). But then, the CRC will be incorrect, and \( B \) will reject the message after decryption! Thus the CRC seems to ensure integrity of the message and to avoid malleability. (This is probably why the designers of WEP added it here.)

Show that the CRC does not increase the security! That is, show how the adversary can modify the ciphertext \( c \) such that \( c \) becomes an encryption of \( m \oplus p \) and such that the CRC within \( c \) is still valid (i.e., it becomes the CRC for \( m \oplus p \)).

Hint: Think of how the \( i \)-th bit of \( \text{CRC}(m \oplus p) \) relates to the \( i \)-th bit of \( \text{CRC}(m) \). (Linearity!)
Solution. For some \( x \), let the \( i \)-th bit of \( CRC(x) \) be denoted \( CRC_i(x) \). Then \( CRC_i(m \oplus p) = \bigoplus_{j \in I_i} (m_j \oplus p_j) = \bigoplus_{j \in I_i} m_j \oplus \bigoplus_{j \in I_i} p_j = CRC_i(m) \oplus CRC_i(p) \). Hence \( CRC(m \oplus p) = CRC(m) \oplus CRC(p) \).

Thus, given a ciphertext \( c = \text{keystream} \oplus (m\|CRC(m)) \), the adversary computes
\[
c' := c \oplus (p\|CRC(p)) = \text{keystream} \oplus ((m \oplus p)\|(CRC(m) \oplus CRC(p)))
= \text{keystream} \oplus ((m \oplus p)\|(CRC(m \oplus p))).
\]

Hence \( c' \) is a valid encryption of \( m \oplus p \) (including a valid CRC checksum).

Problem 3: Birthday attack

Implement a birthday attack for a hash function with 48 bit output. The python code in birthday.py contains template code, fill in the code for the function find_collision.

Solution.

```python
def find_collision():
    # table contains a mapping hash -> preimage, to be filled
    # in the following loop
    table = dict()
    while True:
        x = random.randint(0,2**(hashlen*2))
        h = H(x)
        if h in table:
            return (x,table[h])
        table[h] = x
```
