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: 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.

```python
#!/usr/bin/python3

import sys
if sys.version_info < (3,):
    print("Use Python 3 to run this code")
    exit(1)

import hashlib, random

sha256 = hashlib.new('sha256')
```

1
hashlen = 48

# This is not the right solution. Too slow.
# On my computer:
# hashlen | time
# 16    | 0.1 sec
# 24    | 45 sec
# 32    | 7 hours
# 48    | estimate: 52 years
# 64    | estimate: 3.4 million years
# Of course, this is highly unoptimized code. The same algorithm would run much faster
# if implemented well

def find_collision_slow():
    while True:
        x1 = random.randint(0,2**(hashlen*2))
        x2 = random.randint(0,2**(hashlen*2))
        h1 = H(x1)
        h2 = H(x2)
        if h1==h2 and x1!=x2:
            return (x1,x2)

# Commented out because it’s too slow. You can try it out using smaller values of hashlen
# (x1,x2) = find_collision_slow()
# print (x1,x2)
# assert x1 != x2
# assert H(x1) == H(x2)

# Collision finding using birthday attack
# Returns a pair \((x_1,x_2)\) such that \(H(x_1) = H(x_2)\)
# My code, quite unoptimized (using off-the-shelf python datastructures) takes the following time:
# hashlen | time
# 16 | <0.1 sec
# 24 | <0.1 sec
# 32 | 0.5 sec
# 48 | 72 sec
# 64 | 7min 20sec
def find_collision():
    # Put your code here
    return (1,2)
(x1,x2) = find_collision()
print (x1,x2)
assert x1 != x2
assert H(x1) == H(x2)

Solution.

def find_collision():
    # table contains a mapping hash \(\rightarrow\) 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  

Solution.