Homework 6

April 23, 2022

You will need 50% of all homework points to qualify for the exam.
You may hand in your solutions in person or by email to kristiinesaarmann@gmail.com.
If you submit by email, either scan a handwritten solution or typeset your solution
readably. I do not consider ASCII formulas readable.
When submitting, indicate your name and your matriculation number.
The total number of points for each homework is 20 (not including points for bonus
problems, if available).
For submitting your solution in a nicely typeset way (e.g., using LaTeX), you get up
to 3 bonus points, but not more than 30% of the points you reached for content.

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?

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.

#!/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')

# Change this to something lower for experiments but make sure to put it back to 48 afterwards
# Must be a multiple of 4
hashlen = 48

# A hash function:
# The input is an integer.
# The output is a 'hash_len' bit string, encoded in hex (8 bytes)
def H(number:int) -> str:
    hash = sha256.copy()
    hash.update(str(number).encode('ascii'))
    return hash.hexdigest()[0:hashlen//4]

assert H(123) != H(1230)
assert len(H(1))*4 == hashlen

# 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 muuuch 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 datastructes) takes the following time:
# hashlen | time
# 16 | <0.1 sec
# 24 | <0.1 sec
# 32 | 0.5 sec
# 48 | 72 sec
# 64 | 7 min 20 sec
def find_collision():
    return (1, 2)  # Put your code here
(x1, x2) = find_collision()
print (x1, x2)
assert x1 != x2
assert H(x1) == H(x2)

3 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 \(c := \text{keystream} \oplus (m \parallel CRC(m))\). Here \(\text{keystream}\) is the RC4 keystream computed from \(IV\) and \(k\) (we do not care how).

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