Homework 1

1. Find a collision in each of the hash functions below:

   a. H(x) = x mod 7^{12}, where x can be any integer

   x = 3 and x = 7^{12} +3

   b. H(x) = # of 1-bits in x, where x can be any bit string

   x = 1516 and x =7181

   c. H(x) = the three least significant bits of x, where x can be any bit string

   x = bear and x = pear

2. Prove the statement: In a class of 500 students, there must be two students with the same birthday.

   Proof:

   1) Every student has only one birthday.
   2) There are no more than 366 days per year.
   3) There are 500 students in this class.
   4) Using the pigeon-hole principle, there must be two students with the same birthday.
      a. Pigeonholes: Birthday [Jan.1- Dec.31, no more than 366]
      b. Pigeons: Students [500]
      c. Collision: There must be two students "mapped" to a specific birthday.

3. Find an x such that H (x ∘ id) ∈ Y where

   a) H = SHA-256
   b) Id=0xED00AF5F774E4135E7746419FEB65DE8AE17D6950C95CEC3891070FBB5B03C77
   c) Y is the set of all 256 bit values that have some byte with the value 0x1D.

   Solution:

x=

050FB71499E36E3B7B63586FC99956FAAB26FB71655D10F67334E8D59B218FD9

```java
package assignment1;

import java.io.ByteArrayOutputStream;

public class solution {

    public static void main(String[] args) throws NoSuchAlgorithmException, IOException {
        for (int i = 0; i < 100; i++){
          // Generate a pseudo-random 256-bit nonce.
           byte[] x = new byte[32]; //256 bit array
           new Random().nextBytes(x); //pseudo-random

          //Convert a hex string into a byte array. API requires omitting the leading "0x".
           String hex = "ED00AF5F774E4135E7746419FEB65DE8AE17D6950C95CEC3891070FBB5B03C77";
           byte[] id = DatatypeConverter.parseHexBinary(hex);

          // Concatenate two byte arrays
           ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
           outputStream.write(x);
           outputStream.write(id);
           byte concatHex[] = outputStream.toByteArray();

           MessageDigest digest = MessageDigest.getInstance("SHA-256");
           byte[] hash = digest.digest(concatHex); // SHA256 hash

           for(byte b:hash){
               if(b==0x1D)
                   System.out.println(DatatypeConverter.printHexBinary(x));
           }
        }
    }
}
```

```
<terminated> solution [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_101.jdk/Cor
050FB71499E36E3B7B63586FC99956FAAB26FB71655D10F67334E8D59B218FD9
5FFBCC81E4DB1704A0D591AEDEE550FADEFDE8DCA9D077C0259707DEFF8B8D44
5803F8F2BE5C35DC5BE42CBD72244B6505717C500A65B2893DA819CC556EFD4F
CC29D45230C65D6CD410C6081E4B6BB8A76DA5522D294A3FE7A9D6F5FF8AD6ED
5787A498D474AC6985F125092A2714ED5D927E6E2C1FD5FCB654647BD15C4733
039B83F25E6CB78296AB50F7A06AC3043C8A9CB3D492F45D45635A5648E7928D
84669577A9377E0A2CEAA3A0EC3D367A41376BCBD9D9506D1C0A76D558BB7CFB
A35D2E472955F4EF4E728470447661C566B2453645278AFBDC934F67D3479D9B
795370AE830AD7AC7809E549713A916BA8147A631A1CC3076BB6D5426EB6D29B
13EBF98AA9AFC884D4187A469623D97CB911E02941968CDBFD0C19C13ABB765B
04B3498506E87F0D48E41E587DDE8A01EE1DA5FA56E294DA5F20F72AF5B1D014
F100A77D4B6E900A7A06F3DA2C37C77D908A286DA5B40782D35083F5AB5BC656
40828643F41420E2DCE2A7393317298AFC2EA698632C1DAD9AFB6C6E9F4CD6DF
40828643F41420E2DCE2A7393317298AFC2EA698632C1DAD9AFB6C6E9F4CD6DF
42A2E09B25E2A5D598EB96748653B82651E8D7845653803F2B65253C1D0E85CC
6E46BE69B44E25F9D39C6769BD80B3D90FFE7843B757201C831CA7654A503658
```

4. To address Bob's concern, the main idea of mechanism is to use hash function as a commitment scheme.

   a. Details of mechanism:

      I would like to implement the digital analog of the following physical scheme: Alice and Bob agree on a secure hash function h. Alice chooses a random string r_A and Bob chooses a random string r_B. Bob tells Alice r_B.

      Now, Alice gives a number x between 1 to 10. Alice sends h (x, r_A, r_B) to Bob and asks Bob to guess that number. Let's say Bob guess a number y. Then Alice tells Bob (x, r_A) and they can verify that x=y by checking that h (x, r_A, r_B) = h (y, r_A, r_B). In this way if Bob gives wrong number, then Alice can prove that he was wrong. Obviously, if Bob gives the number correctly, then the two hashes match.

b. Why does it work?

It's extremely hard for Alice to cheat because if Bob says "8" for example when the number was indeed "8" but Alice wants to trick him into thinking it was another number likes"7", she'd have to come up with a random string r such that h (7, r, r_B) = h(8, r_A, r_B), which is hard by the assumption that h is a secure hash function and the fact that Bob chose r_B. Essentially, the purpose of r_A and h are to make Alice "commit" to her initial number x. The point of r_A is so that, without it, Alice might pick some r_A for which she knows another string r which might let her lie.