Problem Set 1

Submit your answers in a simple text file or as a link to a google doc with public access.

1. Find a collision in each of the hash functions below:
    a. H(x) = x mod 512, where x can be any integer
    b. H(x) = number of 0-bits in x, where x can be any bit string
        i. Note: a "bit string" is simply a sequence of 0s and 1s
        ii. For example, 01011011 is a bit-string
    c. H(x) = the five least significant bits of x, where x can be any bit string
        i. Assume the bits in the bit string are ordered in little endian encoding, where the least significant bit is on the left. For example, in the bit string 01010, the least significant two bits is 01.
        ii. Please read the wikipedia article on little endian vs big endian encoding.
        iii. Little endian encoding of the number 4 is 001 and its big endian encoding 100.
        iv. Example:
            1. Suppose the little endian encoding of x is 011010111010110
                a. Then, the five least significant bits of x is 01101
            2. Suppose the little endian encoding of x is 011
                a. Then, the five least significant bits of x is 01100
        v. Two different ways to order the bits in the bit string
            1. Little endian: Least significant bit on the left, and most significant bit on the right
            2. Big endian: Most significant bit on the left, and least significant bit on the right
            3. Little endian and big endian are the reverse of each other
        vi. In the decimal number 539, we use big endian to represent the number in decimal.
        vii. Back to the question, suppose we have function F that, when we give at x, it gives us only the 5 least significant of x.
            1. F(11001011010) = 11001
            2. F(110) = 11000

ANSWER:

a. H(x) = x mod 512:

For a, Collision is x1 = 2 and x2 = 514

b. H(x) = number of 0-bits in

For b, Collision is x1: 1101010 and x2: 1110101

c. Little endian: H(x) = the five least significant bits of x: Collision: 01111 and 11111

Explanation: Both bit strings have the same five least significant bits, which is 01111 in little endian encoding.

2. Implement a program to 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.
    d. Assume SHA-256 is puzzle-friendly. Your answer for x must be in hexadecimal.
    e. Use any language you'd like, however we highly recommend using RUST as it is a critical language in blockchain:
        i. Learn the necessary RUST by reading chapters 1-4 of the Rust Lang book and the Rust By Example reference. Use the following RUST crates: https://crates.io/crates/hex, https://crates.io/crates/sha2, https://crates.io/crates/rand
    f. **Caution**:
        i. The notation "x ◦ id" means the byte array x concatenated with the byte array id. For example, 11110000 ◦ 10101010 is the byte array 1111000010101010.
        ii. The following two code segments are not equivalent:

INCORRECT	**CORRECT**

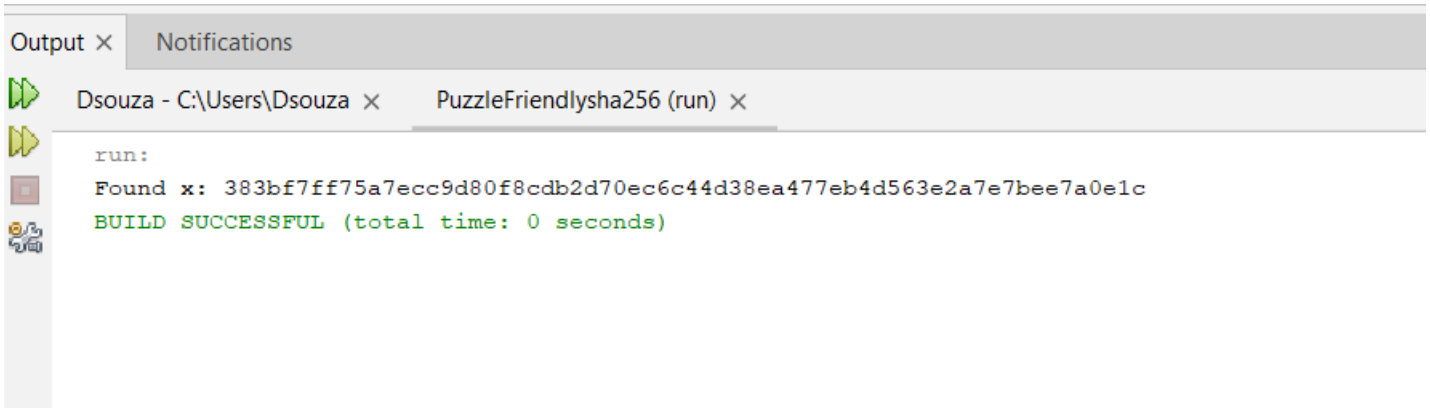
`let id_hex = "1D253A2F";`
 	
`if id_hex.contains("1D")`
`{`
`  return;`
`}`
	`let id_hex = "1D253A2F";`
	
	`let decoded = hex::decode(id_hex).expect("Decoding failed");`
	`let u = u8::from(29); //29 in decimal is 0x1d in hex`
	`if decoded.contains(&u) {`
	`  return;`
	`}`

The second code segment above is the correct way to check whether 0x1D is a byte in 0x1D253A2F. Remember that hex format is only a way to represent a byte sequence in a human readable format. You should never perform operations directly on hex-string representations. Instead, you should first convert hex-strings into byte arrays, then perform operations on the byte arrays directly, and then convert the final byte array into a hex format when giving your answer. Performing operations directly on the hex strings is incorrect.

CODE:

```
package puzzleFriendly;

/**
 *
 * @author Dsouza
 */
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
public class puzzleFriendlysha256 {
   public static void main(String[] args) throws NoSuchAlgorithmException {
   byte[] id = hexStringToByteArray("ED00AF5F774E4135E7746419FEB65DE8AE17D6950C95CEC3891070FBB5B03C77");

   MessageDigest md = MessageDigest.getInstance("SHA-256");
   byte[] x = md.digest(id);

   while (!checkByte(x, (byte) 0xff)) {
      id[15]++;
       x = md.digest(id);
      }
   System.out.println("Found x: " + byteArrayToHexString(x));
   }

   private static boolean checkByte(byte[] bytes, byte value) {
      for(byte b: bytes){
         if(b == value){
            return true;
         }
      }
      return false;
   }

   public static String byteArrayToHexString(byte[] bytes){
      StringBuilder sb = new StringBuilder();
      for(byte b: bytes){
         sb.append(String.format("%02x", b));
      }
      return sb.toString();
   }

   public static byte[] hexStringToByteArray(String hexString){
      int len = hexString.length();
      byte[] data = new byte[len /2];
      for(int i=0;i<len;i+=2){
         data[i /2] = (byte)((Character.digit(hexString.charAt(i),16) << 4) +
               Character.digit(hexString.charAt(i+1),16));
      }
      return data;
   }
}
```

OUTPUT:


Output
Notifications
Dsouza - C:\Users\Dsouza
PuzzleFriendlysha256 (run)
run:
Found x: 383bf7ff75a7ecc9d80f8cdb2d70ec6c44d38ea477eb4d563e2a7e7bee7a0e1c
BUILD SUCCESSFUL (total time: 0 seconds)

3. Alice and Bob want to play a game over SMS text where Alice chooses a number between 1 and 10 in her head, and then Bob tries to guess that number. If Bob guesses correctly, he wins. Otherwise, Alice wins. However, Bob complains that the game isn't fair, because even if Bob guessed correctly, Alice could lie and claim that she chose a different number than what she initially chose. What can Alice do to prove that she didn't change the number she initially chose? Devise a mechanism to address Bob's concern. Provide a detailed explanation of the mechanism and why it works. An answer with insufficient detail will not receive credit. You should only need to use cryptographic hash functions to solve this problem. Keep the solution simple.

ANSWER: Alice can prove that she didn't change the number she initially chose by using a cryptographic hash function. The commitment scheme is a cryptographic protocol that allows Alice to commit to a value without revealing it, and then later reveal the committed value to Bob, proving that she did not change it. Alice can "hash" the number in her head and send the hash to Bob in SMS text.

Here's how the mechanism would work:

1. Alice chooses a number between 1 and 100 in her head and keeps it secret.
2. Alice calculates the SHA-256 hash of the number she chose. Let's call this hash H.
3. Alice sends Bob a message that includes H.
4. Bob guesses a number between 1 and 10 and sends it to Alice.
5. Alice reveals the number she chose and calculates its SHA-256 hash. Let's call this hash G.
6. Alice compares G to H. If they are the same, Alice tells Bob that he has guessed correctly. If they are different, Alice tells Bob that he has guessed incorrectly.

Here's why this mechanism works:

- Cryptographic hash functions like SHA-256 are one-way functions, meaning it is computationally infeasible to find the input that produced a given output (hash). Therefore, if Alice claims to have chosen a different number than the one she initially chose, she would need to find a new number that hashes to the same value as the original hash H. This is extremely difficult and practically impossible to do.
- Bob can verify that Alice didn't change the number she initially chose by comparing the hash of the number. If the hash is the same, Bob can be confident that Alice did not change the number.