INFO 7500 Cryptocurrency/Smart Contract — Homework1

Mibin Zhu / 001424937

Question1. Find a collision in each of the hash functions below:

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

For a, the collision is x1 = 3 and x2 = 7^12 + 3;

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

For b, the collision is x1 = 10101010 and x2 = 11110000;

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

For c, the collision is x1 = 11101 and x2 = 10101.

Question2. Prove the statement: In a class of 500 students, there must be two students with the same birthday.

Answer :

Only 365 days in a year, even the leap year can contain up to 366 days;

There have 500 students in this class, 500 > 366;

Using the pigeon-hole principle, there must be two students with the same birthday, as Pigeonholes [1 - 366] and Pigeons [1 - 500] have the collision.

Question3.

Find an x such that H (x ◦ id) ∈ Ywhere

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.

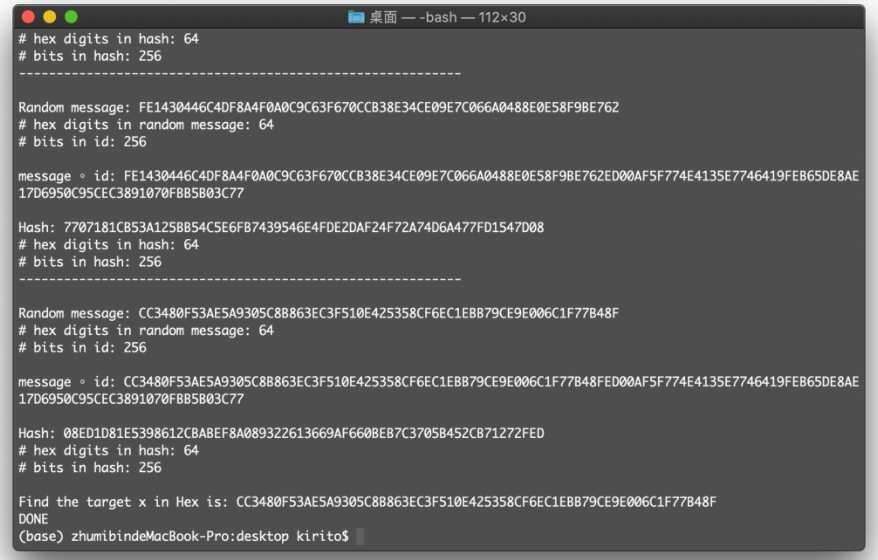
Assume SHA-256 is puzzle-friendly. Your answer for x must be in hexadecimal. You may provide your code for partial credit, if your x value is incorrect. You may use the accompanying CryptoReference1.java file to help you with this question. Submit both your code and your value of x.

```java
import javax.xml.bind.DatatypeConverter;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Random;

public class CryptoReference1
{
    public static void main(String[] args) throws NoSuchAlgorithmException, IOException
    {
        //Convert a hex string into a byte array. API requires omitting the leading "0x".
        String idHex = "ED00AF5F774E4135E7746419FEB65DE8AE17D6950C95CEC3891070FBB5B03C77";
        byte[] id = DatatypeConverter.parseHexBinary(idHex);
        System.out.println();
        System.out.println("Convert a byte array to string in hex: " + DatatypeConverter.printHexBinary(id));

//        String messageStr = "Cryptocurrency is the future";
//        byte[] message = messageStr.getBytes(StandardCharsets.UTF_8);
//        String messageHex = DatatypeConverter.printHexBinary(message);
//        System.out.println("message: " + messageHex);
//        System.out.println("# hex digits in message: " + messageHex.length());
//        System.out.println("# bits in message: " + message.length * 8);
```

```
        while(true)
        {
            System.out.println("------------------------------------------------------------------");
            //Generate a pseudo-random 256-bit message.
            Random ran = new Random();
            byte[] randomMessage = new byte[32];  //256 bit array
            ran.nextBytes(randomMessage);  //pseudo-random
            String randomMessageHex = DatatypeConverter.printHexBinary(randomMessage);
            System.out.println();
            System.out.println("Random message: " + randomMessageHex);
            System.out.println("# hex digits in random message: " + randomMessageHex.length());
            System.out.println("# bits in id: " + randomMessage.length * 8);

            //Concatentate two byte arrays
            ByteArrayOutputStream outputStream = new ByteArrayOutputStream( );
            outputStream.write( randomMessage );
            outputStream.write( id );
            byte concat[] = outputStream.toByteArray( );
            String concatHex = DatatypeConverter.printHexBinary(concat);
            System.out.println();
            System.out.println("message ◦ id: " + concatHex);

            MessageDigest digest = MessageDigest.getInstance("SHA-256");
            byte[] hash = digest.digest(randomMessage);  //SHA256 hash
            String hashHex = DatatypeConverter.printHexBinary(hash);
            System.out.println();
            System.out.println("Hash: " + hashHex);
            System.out.println("# hex digits in hash: " + hashHex.length());
            System.out.println("# bits in hash: " + hash.length * 8);
```

```
            //Check whether the 256 bits hash contians 0x1D.
            boolean flag = false;
            for(byte target : hash)
            {
                if(target == 0x1D)
                {
                    flag = true;
                }
            }

            //Get out of the for loop when we get the target x.
            if(flag == true)
            {
                System.out.println();
                System.out.println("Find the target x in Hex is: " + randomMessageHex);
                break;
            }
        }

        System.out.println("DONE");
    }
}
```

```
桌面 — -bash — 112×30
Last login: Tue Jan 14 14:31:38 on ttys000

The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
(base) zhumibindeMacBook-Pro:~ kirito$ java CryptoReference1
错误：找不到或无法加载主类 CryptoReference1
(base) zhumibindeMacBook-Pro:~ kirito$ cd desktop
(base) zhumibindeMacBook-Pro:desktop kirito$ java CryptoReference1

Convert a byte array to string in hex: ED00AF5F774E4135E7746419FEB65DE8AE17D6950C95CEC3891070FBB5B03C77
------------------------------------------------------------

Random message: 7FED89E3BCC969FF084AEC3C82C70C64F09B42FFB38F4AD76D22E2D6F2A6F300
# hex digits in random message: 64
# bits in id: 256

message ◦ id: 7FED89E3BCC969FF084AEC3C82C70C64F09B42FFB38F4AD76D22E2D6F2A6F300ED00AF5F774E4135E7746419FEB65DE8AE
17D6950C95CEC3891070FBB5B03C77

Hash: 956089EBDD36D81A84C23262E934549E1987B82D5AA27027283418B90AEA318B
# hex digits in hash: 64
# bits in hash: 256
------------------------------------------------------------

Random message: FE1430446C4DF8A4F0A0C9C63F670CCB38E34CE09E7C066A0488E0E58F9BE762
# hex digits in random message: 64
# bits in id: 256

message ◦ id: FE1430446C4DF8A4F0A0C9C63F670CCB38E34CE09E7C066A0488E0E58F9BE762ED00AF5F774E4135E7746419FEB65DE8AE
```



Answer :

By using this code, which is provided by professor. I added a check function to find whether the hash is a 256 bits value containing 0x1D. If it contains, I will break the for loop and show the value of target x, if not, I will run again the random part to get the next target x. For this time of running the code, I got x as :

x = CC3480F53AE5A9305C8B863EC3F510E425358CF6EC1EBB79CE9E006C1F77B48F.

Question4. 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.**

Answer :

To answer this question, we firstly have to find how many problems need to solve.

1) Alice cannot change the number after she decided;
2) No one can change the number that Alice already sent over the SMS;
3) Bob cannot cheat as well; he cannot look at the number Alice already written before he decided.

Depending on these conditions, I will use the asymmetric encryption model to solve this question, as both Alice and Bob will have a primary key and a public key. The public key is open to everyone and the private key is only known by themselves. On the other hand, both Alice and Bob should know how to deal with the hashing function, here is the steps :

1) Both Alice and Bob decide a hashing function H(x);
2) Alice write a number named "Anum", and then use the hashing function H(x) to get the H(Anum);
3) Alice use her own private key "APRIK" to encrypt her hashed number H(Anum), we call this encrypted

information as ENC_{APRIK} (H(Anum)), and she send this to Bob;

4) In order to make Bob have no time to decrypt the ENC_{APRIK} (H(Anum)), he has to send his number "Bnum" to Alice as soon as possible;
5) After Alice receive Bnum, she sends her original number Anum to Bob, also via the SMS;
6) At this time, Bob has the Anum, the ENC_{APRIK} (H(Anum)) and also Alice's public key "APUBK"; So, Bob firstly decrypt ENC_{APRIK} (H(Anum)) by using Alice's public key "APUBK" to get H(Anum) and then use the hashing function H(x) to get Anum, finally compare whether two Anum are the same; If these two Anum are the same, it can proof that Alice didn't change her number;
7) After they ensure this game is valid, they can compare Anum with Bnum, if they are same, then Bob wins, if not, Alice wins.

For the reason why it works, we can view this solution in two aspects. Firstly, Alice sends her number twice, the first time is ENC_{APRIK} (H(Anum)) which can be considered as her digital signature, Bob can decrypt this information by using both Alice's public key "APUBK" and the hashing function H(x) in a quite long time which also ensure Bob have no time to cheat; and the second time is Anum. By comparing whether the number and the "digital signature" are the same, Bob can know whether this game is valid.

Secondly, this asymmetric encryption model also prevents other person change Alice's information via the SMS. Although the "other person" have Alice's public key "APUBK" to decrypt ENC_{APRIK} (H(Anum)) and change the H(Anum), the "other person" don't have Alice private key "APRIK" which do not allow he/she to encrypt this information again. In this way, if Bob cannot decrypt the information sent by Alice using Alice's public key "APUBK", he will know this information is changed and this game is invalid.