Homework 1

Chong Chen

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

1.1 H(x) = $x^2 \bmod 9$, where x can be any integer

Answer: Let $x_1 = 9$ and $x_2 = 18$. Then:

$$H(x_1) = H(x_2) = 0$$

1.2 H(x) = number of 0-bits in x, where x can be any bit string

Answer: Let $x_1 = $ "10011" and $x_2 = $ "010111". Then:

$$H(x_1) = H(x_2) = 2$$

1.3 H(x) = the three least significant bits of x, where x is a 32-bit integer

Answer: Let $x_1 = 1$ and $x_2 = 1 + 2^{30} = 1073741825$. Then:

$$H(x_1) = H(x_2) = 1$$

2 Implement a program to find an x such that $H(x \circ \textbf{id}) \in Y$ where:

1. H = SHA-256
2. id = 0xED00AF5F774E4135E7746419FEB65DE8AE17D6950C95CEC3891070FBB5B03C78
3. Y is the set of all 256-bit values that have some byte with the value $0x2F$
4. Assume SHA-256 is puzzle-friendly.

Your answer for x must be in hexadecimal.

Rust Code [https://github.com/chongchen1999/INFO7500-cryptocurrency/blob/main/hw1/find_x/src/main.rs]:

```rust
use hex;
use rand::Rng;
use sha2::{Digest, Sha256};

fn main() {
    let id_hex_string = "ED00AF5F774E4135E7746419FEB65DE8AE17D6950C95CEC3891070FBB5B03C78";
    const TARGET_BYTE: u8 = 0x2F;

    let id_hex = hex::decode(id_hex_string).expect("Failed to decode hexadecimal string");
    let mut rng = rand::thread_rng();

    loop {
        // Generate a random 32-byte array
        let mut random_bytes = [0u8; 32];
        rng.fill(&mut random_bytes);

        // Concatenate the random bytes and the ID
        let mut concatenated_input = random_bytes.to_vec();
        concatenated_input.extend_from_slice(&id_hex);

        // Compute the SHA-256 hash of the concatenated input
        let hash_result = Sha256::digest(&concatenated_input);

        // Check if the hash contains the target byte
        if hash_result.contains(&TARGET_BYTE) {
            let random_bytes_hex = hex::encode(random_bytes).to_uppercase();
            println!("Found matching x: {}", random_bytes_hex);
            break;
        }
    }
}
```

Dependencies

```toml
[dependencies]
sha2 = "0.10"
rand = "0.8"
hex = "0.4"
```

One Solution:
$x =$ F4D4520CE652631AB2937DE847BA4B539BBA270591A1D30BAFED72E4685890E7

3 Protocol for Playing Rock-Paper-Scissors Over SMS

3.1 Protocol Preparation

Both players need to:

- Agree on a secure hash function, such as SHA-256.
- Have the ability to generate a secret number that cannot be known by others.

3.2 Game Process

Alice's Operations:

1. Alice generates a secret number n_A and selects her gesture (rock, paper, or scissors), denoted as g_A.
2. Combine the secret number n_A and the gesture g_A into a string $s_A = \text{str}(n_A) + \text{str}(g_A)$.
3. Calculate the hash value of the string s_A: $h_A = \text{hash}(s_A)$.
4. Alice sends the hash value h_A to Bob.

Bob's Operations:

1. Bob generates a secret number n_B and selects his gesture g_B.
2. Combine the secret number n_B and the gesture g_B into a string $s_B = \text{str}(n_B) + \text{str}(g_B)$.
3. Calculate the hash value of the string s_B: $h_B = \text{hash}(s_B)$.
4. Bob sends the hash value h_B to Alice.

Exchange of Original Data:

1. Alice sends n_A and g_A to Bob after receiving h_B.
2. Bob sends n_B and g_B to Alice after receiving h_A.

Verification:

1. After receiving n_A and g_A, Bob calculates $s'_A = \text{str}(n_A) + \text{str}(g_A)$ and verifies whether $\text{hash}(s'_A) = h_A$. If they are equal, it confirms Alice has not cheated.
2. Similarly, after receiving n_B and g_B, Alice calculates $s'_B = \text{str}(n_B) + \text{str}(g_B)$ and verifies whether $\text{hash}(s'_B) = h_B$. If they are equal, it confirms Bob has not cheated.

3.3 Result Determination

Both players determine the game result according to the real gestures sent by the other, following the rules of "rock-paper-scissors."

3.4 Anti-Cheating Principle

Each person reveals the secret number and gesture after receiving the hash value from the other person. That means when one person, let's say Alice, receives the secret number and gesture of Bob, her own hash value of the concatenated string of secret number and gesture has already been received by Bob. If she wants to cheat at this point, for example, change g_A to g'_A, she must find a number n'_A such that:

$$\text{Hash}(\text{str}(n'_A) + \text{str}(g'_A)) = \text{Hash}(\text{str}(n_A) + \text{str}(g_A))$$

This is infeasible because of the puzzle-friendliness property of the cryptographic hash function.