Lab 1

Ally Smith (Section A)

April 12, 2022

1. Generating Message Digest and MAC

The easiest thing to notice after generating the different values was that they vary in size. MD5 was 32 digits, or 128 bits. SHA-1 was 40 digits, or 160 bits. SHA-256 was 64 digits, or 256 bits. Other than that, there was no discernable pattern in the hash values.

2. Keyed Hash and HMAC

It is not necessary to use a key of a fixed size for HMAC. This is because the main calculation in an HMAC is provided by any cryptographic hash function, which can take any input size and produce the same output size.

3. The Randomness of One-way Hash

(a) Input file contents:

```
Lorem ipsum dolor sit amet, consectetur adipiscing elit,
sed do eiusmod tempor incididunt ut labore et dolore
magna aliqua.
```

Hash Results:

Hash	**MD5**	**SHA256**
H_1	e9faddf4d13ff132 9e75e2573b86769b	dfee893b95f63da3 8bd4a2e50826bc11 cc9c5b90c7db14ec 1a5ce83db1e90fea
H_2	5ac810b728895819 4ad3282cfbfc4d8c	2cc68a94df19e743 036d5c3f8f88618e f2da49e0c5ac809a c5111ce584e453e6
Shared bits	59 of 128	121 of 256

4. One-Way Property versus Collision-Free Property

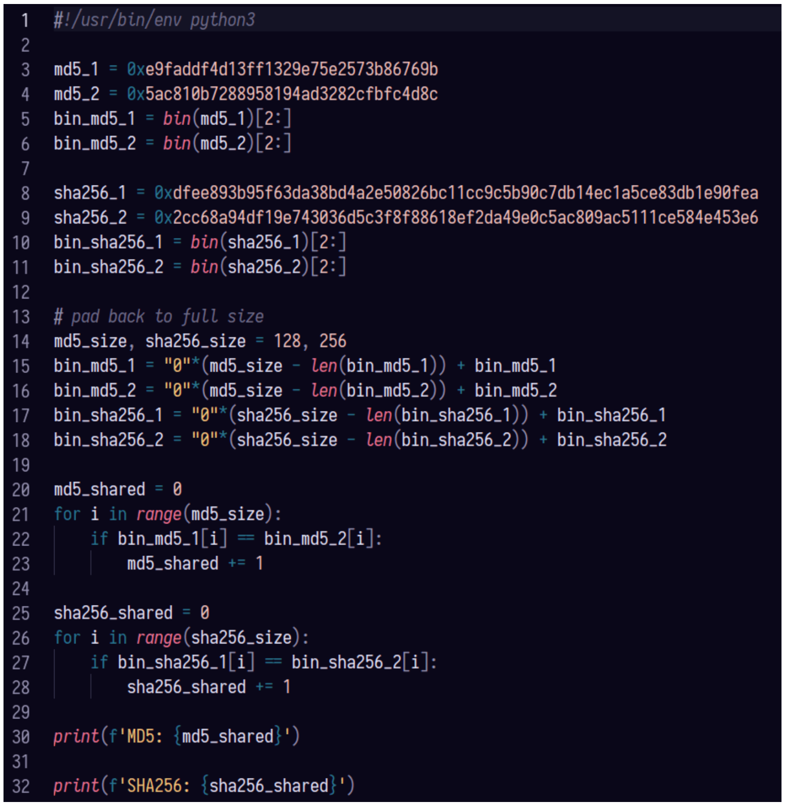
(a) It took an average of 23,129,812 attempts to crack the one-way property

(b) For the collision-free property, it took an average of 28,742 attempts to crack it.

(c) We can assume that it is easier to break the collision-free property based off these test results.

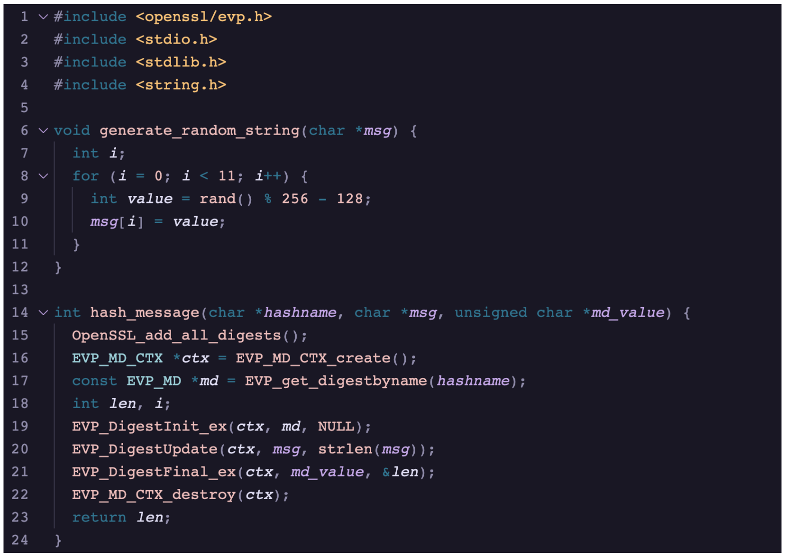
5. Appendix

The script used to count shared bits between H_1 and H_2 for question 3:

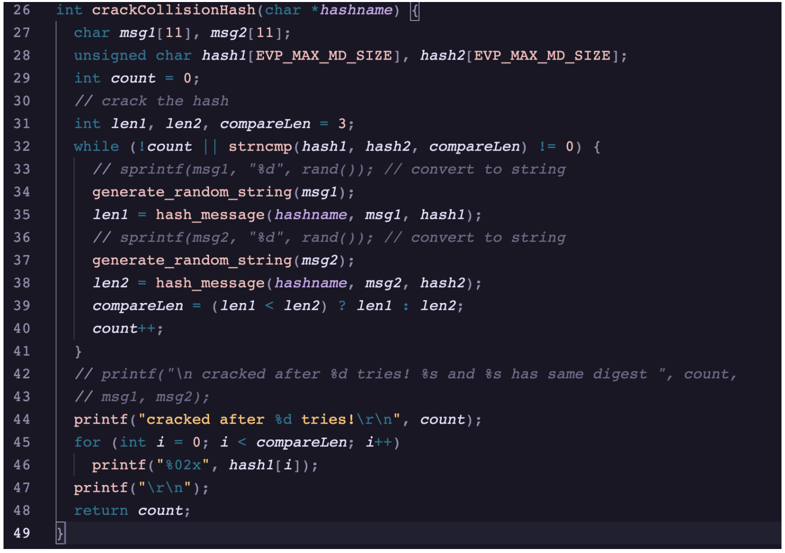


```python
#!/usr/bin/env python3

md5_1 = 0xe9faddf4d13ff1329e75e2573b86769b
md5_2 = 0x5ac810b7288958194ad3282cfbfc4d8c
bin_md5_1 = bin(md5_1)[2:]
bin_md5_2 = bin(md5_2)[2:]

sha256_1 = 0xdfee893b95f63da38bd4a2e50826bc11cc9c5b90c7db14ec1a5ce83db1e90fea
sha256_2 = 0x2cc68a94df19e743036d5c3f8f88618ef2da49e0c5ac809ac5111ce584e453e6
bin_sha256_1 = bin(sha256_1)[2:]
bin_sha256_2 = bin(sha256_2)[2:]

# pad back to full size
md5_size, sha256_size = 128, 256
bin_md5_1 = "0"*(md5_size - len(bin_md5_1)) + bin_md5_1
bin_md5_2 = "0"*(md5_size - len(bin_md5_2)) + bin_md5_2
bin_sha256_1 = "0"*(sha256_size - len(bin_sha256_1)) + bin_sha256_1
bin_sha256_2 = "0"*(sha256_size - len(bin_sha256_2)) + bin_sha256_2

md5_shared = 0
for i in range(md5_size):
    if bin_md5_1[i] == bin_md5_2[i]:
        md5_shared += 1

sha256_shared = 0
for i in range(sha256_size):
    if bin_sha256_1[i] == bin_sha256_2[i]:
        sha256_shared += 1

print(f'MD5: {md5_shared}')

print(f'SHA256: {sha256_shared}')
```

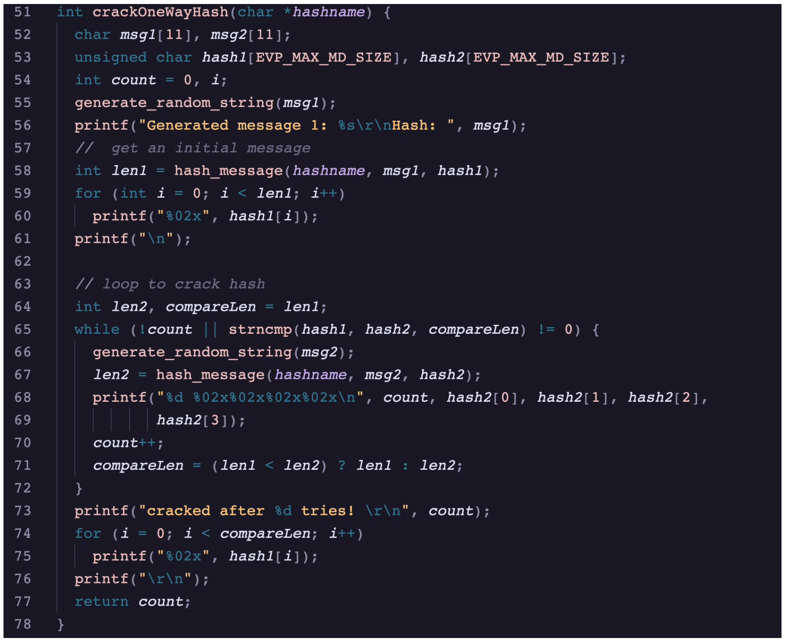
The code I used to complete task 4:



```c
#include <openssl/evp.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void generate_random_string(char *msg) {
    int i;
    for (i = 0; i < 11; i++) {
        int value = rand() % 256 - 128;
        msg[i] = value;
    }
}

int hash_message(char *hashname, char *msg, unsigned char *md_value) {
    OpenSSL_add_all_digests();
    EVP_MD_CTX *ctx = EVP_MD_CTX_create();
    const EVP_MD *md = EVP_get_digestbyname(hashname);
    int len, i;
    EVP_DigestInit_ex(ctx, md, NULL);
    EVP_DigestUpdate(ctx, msg, strlen(msg));
    EVP_DigestFinal_ex(ctx, md_value, &len);
    EVP_MD_CTX_destroy(ctx);
    return len;
}
```



```c
int crackCollisionHash(char *hashname) {
    char msg1[11], msg2[11];
    unsigned char hash1[EVP_MAX_MD_SIZE], hash2[EVP_MAX_MD_SIZE];
    int count = 0;
    // crack the hash
    int len1, len2, compareLen = 3;
    while (!count || strncmp(hash1, hash2, compareLen) != 0) {
        // sprintf(msg1, "%d", rand()); // convert to string
        generate_random_string(msg1);
        len1 = hash_message(hashname, msg1, hash1);
        // sprintf(msg2, "%d", rand()); // convert to string
        generate_random_string(msg2);
        len2 = hash_message(hashname, msg2, hash2);
        compareLen = (len1 < len2) ? len1 : len2;
        count++;
    }
    // printf("\n cracked after %d tries! %s and %s has same digest ", count,
    // msg1, msg2);
    printf("cracked after %d tries!\r\n", count);
    for (int i = 0; i < compareLen; i++)
        printf("%02x", hash1[i]);
    printf("\r\n");
    return count;
}
```



```c
int crackOneWayHash(char *hashname) {
  char msg1[11], msg2[11];
  unsigned char hash1[EVP_MAX_MD_SIZE], hash2[EVP_MAX_MD_SIZE];
  int count = 0, i;
  generate_random_string(msg1);
  printf("Generated message 1: %s\r\nHash: ", msg1);
  //  get an initial message
  int len1 = hash_message(hashname, msg1, hash1);
  for (int i = 0; i < len1; i++)
    printf("%02x", hash1[i]);
  printf("\n");

  // loop to crack hash
  int len2, compareLen = len1;
  while (!count || strncmp(hash1, hash2, compareLen) != 0) {
    generate_random_string(msg2);
    len2 = hash_message(hashname, msg2, hash2);
    printf("%d %02x%02x%02x%02x\n", count, hash2[0], hash2[1], hash2[2],
           hash2[3]);
    count++;
    compareLen = (len1 < len2) ? len1 : len2;
  }
  printf("cracked after %d tries! \r\n", count);
  for (i = 0; i < compareLen; i++)
    printf("%02x", hash1[i]);
  printf("\r\n");
  return count;
}
```