

## Lab 8 Crypto Lab II – One-way Hash Function and MAC

### Task 1, Generating Message Digest and MAC

```
Terminal
[03/24/2017 08:19] seed@ubuntu:~/CSE644/Lab8_code$ openssl dgst -md5 task1.txt
MD5(task1.txt)= 0260d83354492ece3a815f57b32a49c3
[03/24/2017 08:19] seed@ubuntu:~/CSE644/Lab8_code$ openssl dgst -sha1 task1.txt
SHA1(task1.txt)= 7b654abe66cfa10225d2db9dffb40e54272e785b
[03/24/2017 08:19] seed@ubuntu:~/CSE644/Lab8_code$ openssl dgst -sha256 task1.txt
SHA256(task1.txt)= cd335d55361cab825784321aea7c6da60f57b9568c3b01a26e45c525bb1dc01
[03/24/2017 08:19] seed@ubuntu:~/CSE644/Lab8_code$ openssl dgst -md5 task1.txt
MD5(task1.txt)= 0260d83354492ece3a815f57b32a49c3
[03/24/2017 08:27] seed@ubuntu:~/CSE644/Lab8_code$ openssl dgst -sha1 task1.txt
SHA1(task1.txt)= 7b654abe66cfa10225d2db9dffb40e54272e785b
[03/24/2017 08:27] seed@ubuntu:~/CSE644/Lab8_code$ openssl dgst -sha256 task1.txt
SHA256(task1.txt)= cd335d55361cab825784321aea7c6da60f57b9568c3b01a26e45c525bb1dc01
[03/24/2017 08:27] seed@ubuntu:~/CSE644/Lab8_code$ █
```

**Observation:** The three algorithms I used are MD5, SHA1 and SHA256. As we can see in this pic, the length of the generated text is different. MD5 is 16 bytes, SHA1 is 20 bytes and SHA256 is 32 bytes. The same plain text will generate the same hash value no matter how many times you try.

### Task 2, Keyed Hash and HMAC

```
[03/24/2017 08:29] seed@ubuntu:~/CSE644/Lab8_code$ openssl dgst -md5 -hmac "abcdefg" task1.txt
HMAC-MD5(task1.txt)= 54006fd84803b0a2341142b9442bdb27
[03/24/2017 08:29] seed@ubuntu:~/CSE644/Lab8_code$ clear

[03/24/2017 08:29] seed@ubuntu:~/CSE644/Lab8_code$ openssl dgst -md5 -hmac "abcdefg" task1.txt
HMAC-MD5(task1.txt)= 54006fd84803b0a2341142b9442bdb27
[03/24/2017 08:29] seed@ubuntu:~/CSE644/Lab8_code$ openssl dgst -sha1 -hmac "abcdefg" task1.txt
HMAC-SHA1(task1.txt)= a150112a8b17e0c59f8354b180bc7051e04d661f
[03/24/2017 08:39] seed@ubuntu:~/CSE644/Lab8_code$ openssl dgst -sha256 -hmac "abcdefg" task1.txt
HMAC-SHA256(task1.txt)= 506273a009b3defeae26f4b47dd645177844153f23cadb537cfb6bfe25d22c81
[03/24/2017 08:39] seed@ubuntu:~/CSE644/Lab8_code$ openssl dgst -md5 -hmac "abcdefgggggggggggggg" task1.txt
HMAC-MD5(task1.txt)= e950eee8998348da84ab4e440d6285ff
[03/24/2017 08:40] seed@ubuntu:~/CSE644/Lab8_code$ openssl dgst -md5 -hmac "abcdefgggggggggggggggankjnkjnlkj" task1.txt
HMAC-MD5(task1.txt)= b7302b39a3e1b63b02d97c0b5ac40ca0
[03/24/2017 08:40] seed@ubuntu:~/CSE644/Lab8_code$ openssl dgst -sha1 -hmac "abcdefggggggggggggggg" task1.txt
HMAC-SHA1(task1.txt)= aa9df03b27ffc3621875b850f122b37d4c3f1f4d
[03/24/2017 08:40] seed@ubuntu:~/CSE644/Lab8_code$ openssl dgst -sha1 -hmac "abcdefgggggggggggggggankjnkjnlkj" task1.txt
HMAC-SHA1(task1.txt)= 8db7adf21e188d283881fe32a92bf99e837f4be7
[03/24/2017 08:40] seed@ubuntu:~/CSE644/Lab8_code$ openssl dgst -sha256 -hmac "abcdefggggggggggggggg" task1.txt
HMAC-SHA256(task1.txt)= 20c71a6de924450a4e632bf316d8db58dc170b8613277a374022625a0a338e59
[03/24/2017 08:40] seed@ubuntu:~/CSE644/Lab8_code$ openssl dgst -sha256 -hmac "abcdefgggggggggggggggankjnkjnlkj" task1.txt
HMAC-SHA256(task1.txt)= 8fac76dbb181427c1ded27497cab973766be8823f5c15497474ed325e5d95dd6
[03/24/2017 08:41] seed@ubuntu:~/CSE644/Lab8_code$ █
```

**Observation:** If the hash algorithm is fixed, then no matter how long the HMAC key is, the length of hash value is fixed. However, if the HMAC key is different, the generated text is different, even though the plain text is the same.

#### • Do we have to use a key with a fixed size in HMAC? WHY?

It is unnecessary to use a key with a fixed size. Because if the HMAC key is longer than the block length of a specific hash algorithm, the extra part will be cut; if the HMAC key is shorter than the block size, zero will be padded.

### Task 3, The Randomness of One-Way Hash

```
Terminal
[03/25/2017 19:13] seed@ubuntu:~/CSE644/Lab8_code$ ls -al task3.txt
-rw-rw-r-- 1 seed seed 39 Mar 25 19:13 task3.txt
[03/25/2017 19:13] seed@ubuntu:~/CSE644/Lab8_code$ cat task3.txt
This is a plain text for test purpose.
[03/25/2017 19:13] seed@ubuntu:~/CSE644/Lab8_code$ openssl dgst -md5 task3.txt
MD5(task3.txt)= 24c38e02c947a84ca296b4d953e1070a
[03/25/2017 19:14] seed@ubuntu:~/CSE644/Lab8_code$ openssl dgst -sha256 task3.txt
SHA256(task3.txt)= 85b91b2ed2ecb407adae5ded0e09aa9cd7cea48676ad38eae6d20b450273896d
[03/25/2017 19:14] seed@ubuntu:~/CSE644/Lab8_code$ bless task3.txt
Directory '/home/seed/.config/bless/plugins' not found.
Directory '/home/seed/.config/bless/plugins' not found.
Directory '/home/seed/.config/bless/plugins' not found.
Could not find file "/home/seed/.config/bless/export_patterns".
[03/25/2017 19:14] seed@ubuntu:~/CSE644/Lab8_code$ cat task3.txt
Uhis is a plain text for test purpose.
[03/25/2017 19:14] seed@ubuntu:~/CSE644/Lab8_code$ openssl dgst -md5 task3.txt
MD5(task3.txt)= a16e6737c5f3bc350bb60393c36285eb
[03/25/2017 19:14] seed@ubuntu:~/CSE644/Lab8_code$ openssl dgst -sha256 task3.txt
SHA256(task3.txt)= b86cd2cbe4f75423c47521781adaddfe642f8fee6a1cccc23ae8e1fdd38f2423
[03/25/2017 19:14] seed@ubuntu:~/CSE644/Lab8_code$ ./compare
00100100110000111000111000000010110010010100011110101000010011001010001010010110101101001
101100101010011111000010000011100001010
10100001011011100110011100110111100010111100111011110000101010000101110110110000000111
0010011110000101100010100001011101011
Using MD5, There are 73 bits are same
10000101101110010001101100101110110100101110110010110100000001111010110110101110010111011
1101101000011100000100110101010100111001101011110011101010010010000110011101101010110100
111000111010101110011011010010000010110100010100000010011100111000100101101101
101110000110110011010010100101111001001111011101010100001000111100010001110101001000010
1111000000110101101101101110111111001100100001011111000111111101110011010100001110011
00110011000010001110101110100011100001111110111010011100011110010010000100011
Using SHA256, There are 122 bits are same
[03/25/2017 19:14] seed@ubuntu:~/CSE644/Lab8_code$
```

**Observation:** I used “bless” to flip the fourth bit of the plain text, as we can see, the hash value of the revised text is not quite similar, for both MD5 algorithm and SHA256 algorithm. It seems like generally half of the digits is same if one bit is flipped.

### Task 4, One-Way Property versus Collision-Free Property

- break one-way property

**Design:** 1, supplying a plain text and generating its hash value; 2, randomly generating a string, comparing the first 24 bits of its hash value with the plain text’s; 3, if found matches, report the string.

```
[03/25/2017 14:11] seed@ubuntu:~/CSE644/Lab8_code$ ./one-way a
plain text: a
Hex value of the first 24 bits: 0cc175
Tried 37 times
Random String: a
The first 24 bits hash value of this string:0cc175
[03/25/2017 14:12] seed@ubuntu:~/CSE644/Lab8_code$ ./one-way ab
plain text: ab
Hex value of the first 24 bits: 187ef4
Tried 2355 times
Random String: ab
The first 24 bits hash value of this string:187ef4
[03/25/2017 14:12] seed@ubuntu:~/CSE644/Lab8_code$ ./one-way aaaa
plain text: aaaa
Hex value of the first 24 bits: 74b873
Tried 571125 times
Random String: g0Q2
The first 24 bits hash value of this string:74b873
[03/25/2017 14:13] seed@ubuntu:~/CSE644/Lab8_code$
```

```
[03/25/2017 13:50] seed@ubuntu:~/CSE644/Lab8_code$ ./one-way test
plain text: test
Hex value of the first 24 bits: 098f6b
Tried 4857624 times
Random String: A0hK
The first 24 bits hash value of this string:098f6b
[03/25/2017 13:51] seed@ubuntu:~/CSE644/Lab8_code$ ./one-way hola
plain text: hola
Hex value of the first 24 bits: 4d1863
Tried 8622333 times
Random String: hola
The first 24 bits hash value of this string:4d1863
[03/25/2017 13:53] seed@ubuntu:~/CSE644/Lab8_code$ ./one-way mama
plain text: mama
Hex value of the first 24 bits: eeafbf
Tried 8628987 times
Random String: mama
The first 24 bits hash value of this string:eeafbf
[03/25/2017 13:55] seed@ubuntu:~/CSE644/Lab8_code$ ./one-way world
plain text: world
Hex value of the first 24 bits: 7d7930
Tried 17285929 times
Random String: rZhB1
The first 24 bits hash value of this string:7d7930
[03/25/2017 13:59] seed@ubuntu:~/CSE644/Lab8_code$ ./one-way excl
plain text: excl
Hex value of the first 24 bits: 2afdc5
Tried 11402649 times
Random String: excl
The first 24 bits hash value of this string:2afdc5
[03/25/2017 14:02] seed@ubuntu:~/CSE644/Lab8_code$
```

**Observation:** As we can see in the screenshot above, I have tried 8 times. By using my method of brute-force, the simpler the plaintext, the less the trial times. For the string with four characters, the average running times are around 8,000,000. For these 8 experiments, the average trials are more than 4,000,000.

- **break collision-free property**

**Design:** The basic idea to break collision-free property is: starting from a random number and then increasing this random number by 1 for several times until finding two numbers in this range—from start number to current number—that the first 24 bits have the same hash value.

```
Terminal
[03/29/2017 19:08] seed@ubuntu:~/CSE644/Lab8_code$ ./collision-free
Generated a random number: 340059
The first 24 bits hash value of the random number:385438
Tried 2766 times
The number has the same hash value: 339703
The first 24 bits hash value of this number:385438
[03/29/2017 19:08] seed@ubuntu:~/CSE644/Lab8_code$ ./collision-free
Generated a random number: 402033
The first 24 bits hash value of the random number:67bff8
Tried 5871 times
The number has the same hash value: 396893
The first 24 bits hash value of this number:67bff8
[03/29/2017 19:08] seed@ubuntu:~/CSE644/Lab8_code$ ./collision-free
Generated a random number: 999325
The first 24 bits hash value of the random number:a73acf
Tried 10899 times
The number has the same hash value: 995693
The first 24 bits hash value of this number:a73acf
[03/29/2017 19:08] seed@ubuntu:~/CSE644/Lab8_code$ ./collision-free
Generated a random number: 148882
The first 24 bits hash value of the random number:631f6a
Tried 1695 times
The number has the same hash value: 148397
The first 24 bits hash value of this number:631f6a
[03/29/2017 19:08] seed@ubuntu:~/CSE644/Lab8_code$ ./collision-free
Generated a random number: 896633
The first 24 bits hash value of the random number:378e1e
Tried 6456 times
The number has the same hash value: 892156
The first 24 bits hash value of this number:378e1e
[03/29/2017 19:09] seed@ubuntu:~/CSE644/Lab8_code$ █
```

**Observation:** Using this method to break collision free property, the average trials I observed through the above experiment is around 5000 times. Comparing with the one-way property broken experiment, the trying times is much smaller.

- **Based on your observation, which property is easier to break using the brute-force methods?**

Collision-free property is easier to break based on my observation.

- **Can you explain the difference in your observation mathematically?**

For a hash function with a  $n$ -bit output, it will cost roughly  $2^n$  to break one-way property, and  $2^{n/2}$  to break collision-free property.

## Appendix:

### Compare.c

```
#include <stdio.h>
#include <string.h>

// convert hex to bit
void HexToBit(char* cvrt, char* hexStr)
{
    for(int i=0; i<strlen(hexStr);i++)
    {
        switch(hexStr[i])
```



```
{
    case '0':
        strcat(cvrt, "0000");
        break;
    case '1':
        strcat(cvrt, "0001");
        break;
    case '2':
        strcat(cvrt, "0010");
        break;
    case '3':
        strcat(cvrt, "0011");
        break;
    case '4':
        strcat(cvrt, "0100");
        break;
    case '5':
        strcat(cvrt, "0101");
        break;
    case '6':
        strcat(cvrt, "0110");
        break;
    case '7':
        strcat(cvrt, "0111");
        break;
    case '8':
        strcat(cvrt, "1000");
        break;
    case '9':
        strcat(cvrt, "1001");
        break;
    case 'a':
        strcat(cvrt, "1010");
        break;
    case 'b':
        strcat(cvrt, "1011");
        break;
    case 'c':
        strcat(cvrt, "1100");
        break;
    case 'd':
        strcat(cvrt, "1101");
        break;
    case 'e':
```

```

        strcat(cvrt, "1110");
        break;
    case 'f':
        strcat(cvrt, "1111");
        break;
    default:
        break;
    }
}

// do comparasion
int compareMD5(char* Binary1, char* Binary2)
{
    int cnt=0;
    for(int i=0;i <128; i++)
    {
        if(Binary1[i]==Binary2[i])
            cnt++;
    }
    return cnt;
}

int compareSHA256(char* Binary1, char* Binary2)
{
    int cnt=0;
    for(int i=0;i <256; i++)
    {
        if(Binary1[i]==Binary2[i])
            cnt++;
    }
    return cnt;
}

int main()
{
    char* txt1="24c38e02c947a84ca296b4d953e1070a";
    char* txt2="a16e6737c5f3bc350bb60393c36285eb";
    char cvrt1[1024]={};char cvrt2[1024]={};
    HexToBit(cvrt1,txt1); HexToBit(cvrt2,txt2);
    printf("%s\n%s\n", cvrt1, cvrt2);
    int cnt=compareMD5(cvrt1,cvrt2);
    printf("Using MD5, There are %d bits are same\n", cnt);
    ////////////////////////////////////
    char* sha1="85b91b2ed2ecb407adae5ded0e09aa9cd7cea48676ad38eae6d20b450273896d";

```

```

char* sha2="b86cd2cbe4f75423c47521781adaddfe642f8fee6a1cccc23ae8e1fdd38f2423";
char cvrt3[1024]={};char cvrt4[1024]={};
HexToBit(cvrt3,sha1); HexToBit(cvrt4,sha2);
printf("%s\n%s\n", cvrt3, cvrt4);
cnt=compareSHA256(cvrt3,cvrt4);
printf("Using SHA256, There are %d bits are same\n", cnt);
return 0;
}

```

### One-way.c

```

#include <stdio.h>
#include <openssl/evp.h>
#include <string.h>
void GetHashValue(char* plain, unsigned char* hashValue)
{
    EVP_MD_CTX* mdctx;
    const EVP_MD* md;
    unsigned char md_value[EVP_MAX_MD_SIZE];
    int md_len, i;
    OpenSSL_add_all_digests();

    md=EVP_get_digestbyname("md5");

    // allocate, initialize and return a digest context
    mdctx=EVP_MD_CTX_create();
    // set up digest context ctx to use a digest type
    EVP_DigestInit_ex(mdctx,md,NULL);
    // hashes cnt bytes of data at plain into the digest context ctx
    EVP_DigestUpdate(mdctx, plain, strlen(plain));
    // retrieves the digest value from ctx and places it in md
    EVP_DigestFinal_ex(mdctx, hashValue, &md_len);
    // cleans up digest context ctx and frees up the space allocated to it
    EVP_MD_CTX_destroy(mdctx);
    EVP_cleanup();
}

int main(int argc, char* argv[])
{
    char* plain=argv[1];
    unsigned char hashValue[EVP_MAX_MD_SIZE];
    unsigned char dict[]="0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz";
    char str[1000];
    GetHashValue(plain, hashValue);
    printf("plain text: %s\n", plain);
}

```

```

// print the first 24 bits of the hash value
printf("Hex value of the first 24 bits: ");
for(int i=0;i<3;i++) printf("%02x", hashValue[i]);
printf("\n");
// generate random strings for hash

int cnt=0, j=0,k=0;
while(1)
{
    unsigned char temp[EVP_MAX_MD_SIZE];
    j=cnt; k=0;
    while(j>=1)
    {
        str[k]=dict[j%(strlen(dict)-1)];
        k+=1;
        j=j/strlen(dict);
    }
    GetHashValue(str,temp);
    if(strncmp(temp, hashValue, 3)==0)
    {
        printf("Tried %d times\n", cnt+1);
        printf("Random String: %s\n", str);
        printf("The first 24 bits hash value of this string:");
        for(int i=0;i<3;i++)printf("%02x",temp[i]);
        //for(int i=0;i<3;i++)printf("%02x",hashValue[i]);
        printf("\n");
        break;
    }
    cnt++;
}
return 0;
}

```

### **Collision-free.c**

```

#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>
#include <openssl/evp.h>
#include <string.h>

void HashValue(char* plain, unsigned char* hashValue)
{
    EVP_MD_CTX* mdctx;
    const EVP_MD* md;

```



```

int md_len, i;
OpenSSL_add_all_digests();

md=EVP_get_digestbyname("md5");

// allocate, initialize and return a digest context
mdctx=EVP_MD_CTX_create();
// set up digest context ctx to use a digest type
EVP_DigestInit_ex(mdctx,md,NULL);
// hashes cnt bytes of data at plain into the digest context ctx
EVP_DigestUpdate(mdctx, plain, strlen(plain));
// retrieves the digest value from ctx and places it in md
EVP_DigestFinal_ex(mdctx, hashValue, &md_len);
// cleans up digest context ctx and frees up the space allocated to it
EVP_MD_CTX_destroy(mdctx);
EVP_cleanup();
}

int main(int argc, const char* argv)
{
    unsigned char hashValue[EVP_MAX_MD_SIZE];
    srand((unsigned)time(NULL));
    int random=rand()%1000000+1;
    int begin=random;
    unsigned char* hash=(unsigned char*)(malloc(sizeof(unsigned char)*1000000));
    // convert to string to get hash value
    char* plain=malloc(16);
    snprintf(plain, 16, "%d", random);
    // get hash value
    HashValue(plain, hashValue);

    memcpy(hash, hashValue,3);

    // generate random strings for hash
    int cnt=1, j=3,k=0;
    while(1)
    {
        random+=1;
        snprintf(plain, 16, "%d", random);
        // get hash value
        HashValue(plain, hashValue);

        for(int i=0;i<j;i=i+3)
        {

```

```

        if(memcmp(hash+i, hashValue, 3)==0)
        {
            printf("Genarated a random number: %d\n", random);
            // print the first 24 bits
            printf("The first 24 bits hash value of the random number:");
            for(int p=0;p<3;p++)printf("%02x",(hash+i)[p]);
            printf("\n");
            //printf("H1: %s, H2: %s\n", hash+i, hashValue);
            printf("Tried %d times\n", cnt);
            printf("The number has the same hash value: %d\n", begin+i/3);
            printf("The first 24 bits hash value of this number:");
            for(int z=0;z<3;z++)printf("%02x",hashValue[z]);
            printf("\n");
            return 0;
        }
    }
    memcpy(hash+j, hashValue, 3);
    j+=3;
    cnt++;
}
}

```