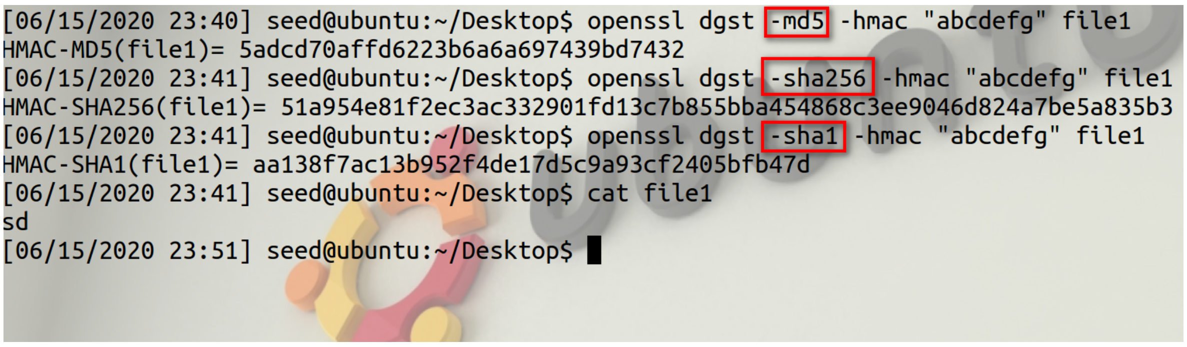
2020 Spring 验证哈希函数的性质

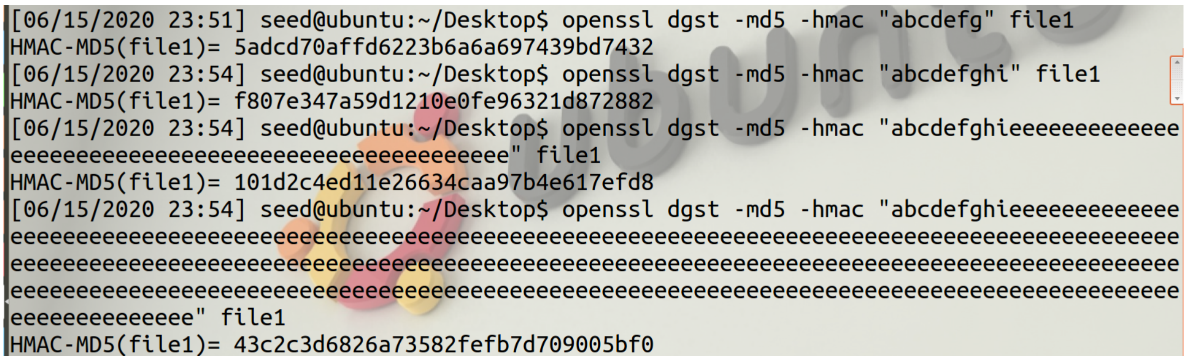
Task 1: Generating Message Digest and MAC



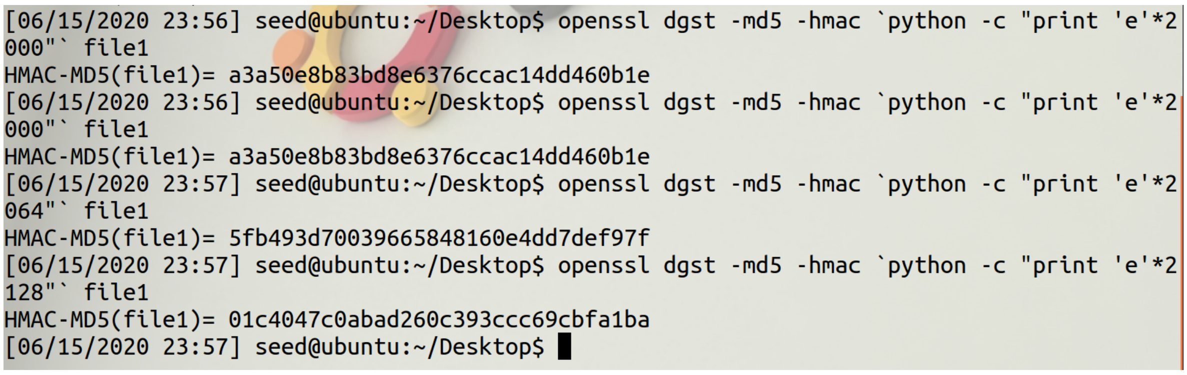
```
[06/15/2020 23:40] seed@ubuntu:~/Desktop$ openssl dgst -md5 -hmac "abcdefg" file1
HMAC-MD5(file1)= 5adcd70affd6223b6a6a697439bd7432
[06/15/2020 23:41] seed@ubuntu:~/Desktop$ openssl dgst -sha256 -hmac "abcdefg" file1
HMAC-SHA256(file1)= 51a954e81f2ec3ac332901fd13c7b855bba454868c3ee9046d824a7be5a835b3
[06/15/2020 23:41] seed@ubuntu:~/Desktop$ openssl dgst -sha1 -hmac "abcdefg" file1
HMAC-SHA1(file1)= aa138f7ac13b952f4de17d5c9a93cf2405bfb47d
[06/15/2020 23:41] seed@ubuntu:~/Desktop$ cat file1
sd
[06/15/2020 23:51] seed@ubuntu:~/Desktop$ 
```

md5、sha256、sha1 函数生成hash值的位数并不相同

Task 2: Keyed Hash and HMAC



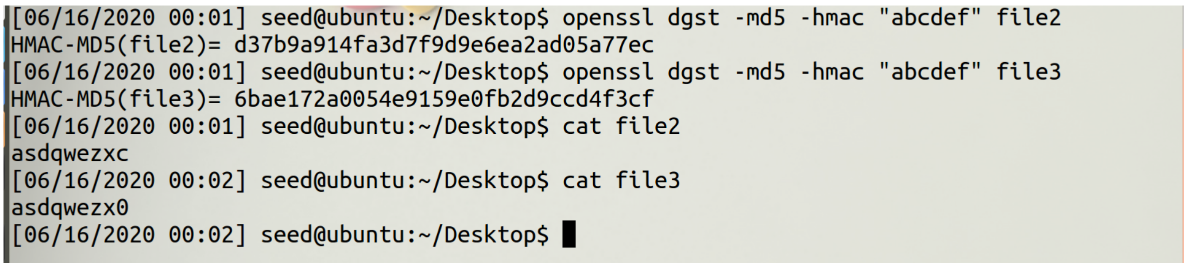
```
[06/15/2020 23:51] seed@ubuntu:~/Desktop$ openssl dgst -md5 -hmac "abcdefg" file1
HMAC-MD5(file1)= 5adcd70affd6223b6a6a697439bd7432
[06/15/2020 23:54] seed@ubuntu:~/Desktop$ openssl dgst -md5 -hmac "abcdefghi" file1
HMAC-MD5(file1)= f807e347a59d1210e0fe96321d872882
[06/15/2020 23:54] seed@ubuntu:~/Desktop$ openssl dgst -md5 -hmac "abcdefghieeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee" file1
HMAC-MD5(file1)= 101d2c4ed11e26634caa97b4e617efd8
[06/15/2020 23:54] seed@ubuntu:~/Desktop$ openssl dgst -md5 -hmac "abcdefghieeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeee" file1
HMAC-MD5(file1)= 43c2c3d6826a73582fefb7d709005bf0
```



```
[06/15/2020 23:56] seed@ubuntu:~/Desktop$ openssl dgst -md5 -hmac `python -c "print 'e'*2
000"` file1
HMAC-MD5(file1)= a3a50e8b83bd8e6376ccac14dd460b1e
[06/15/2020 23:56] seed@ubuntu:~/Desktop$ openssl dgst -md5 -hmac `python -c "print 'e'*2
000"` file1
HMAC-MD5(file1)= a3a50e8b83bd8e6376ccac14dd460b1e
[06/15/2020 23:57] seed@ubuntu:~/Desktop$ openssl dgst -md5 -hmac `python -c "print 'e'*2
064"` file1
HMAC-MD5(file1)= 5fb493d70039665848160e4dd7def97f
[06/15/2020 23:57] seed@ubuntu:~/Desktop$ openssl dgst -md5 -hmac `python -c "print 'e'*2
128"` file1
HMAC-MD5(file1)= 01c4047c0abad260c393ccc69cbfa1ba
[06/15/2020 23:57] seed@ubuntu:~/Desktop$ 
```

如图，可以是无限长。

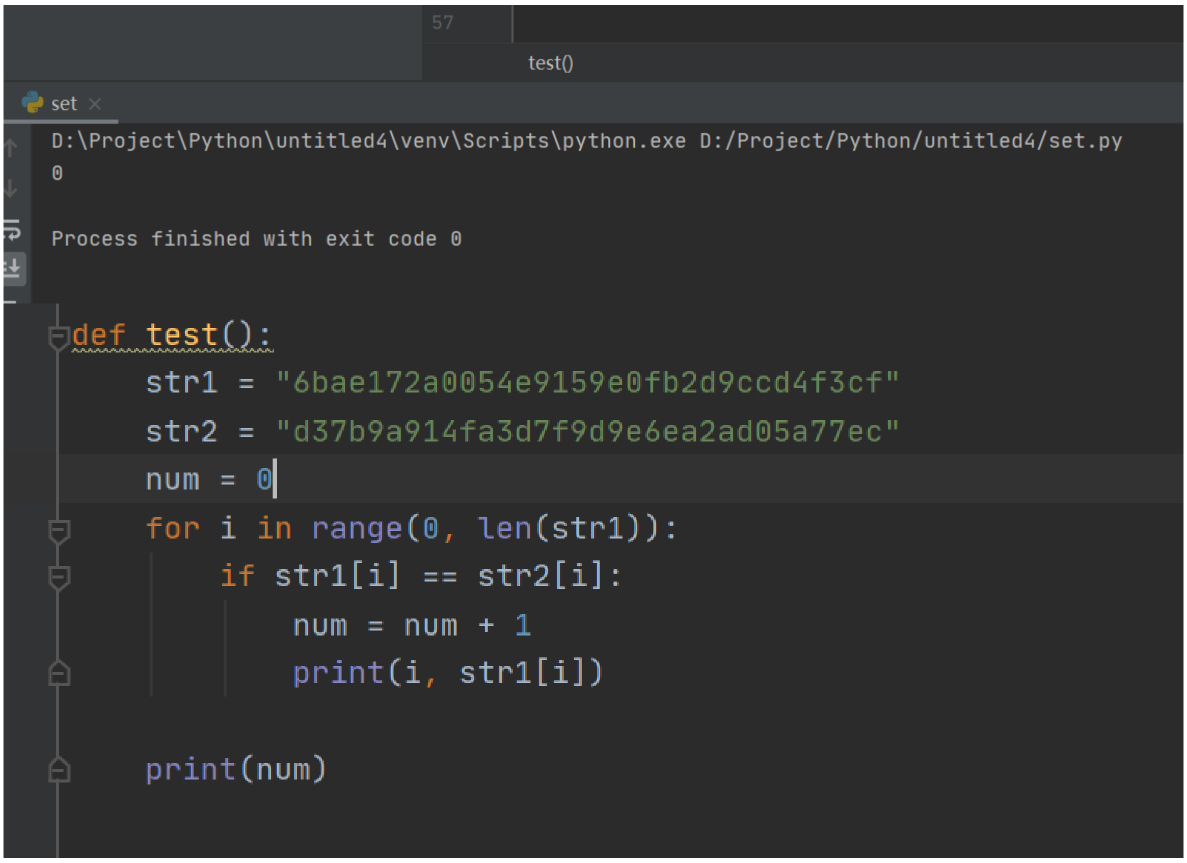
Task 3: The Randomness of One-way Hash



```
[06/16/2020 00:01] seed@ubuntu:~/Desktop$ openssl dgst -md5 -hmac "abcdef" file2
HMAC-MD5(file2)= d37b9a914fa3d7f9d9e6ea2ad05a77ec
[06/16/2020 00:01] seed@ubuntu:~/Desktop$ openssl dgst -md5 -hmac "abcdef" file3
HMAC-MD5(file3)= 6bae172a0054e9159e0fb2d9ccd4f3cf
[06/16/2020 00:01] seed@ubuntu:~/Desktop$ cat file2
asdqwezxc
[06/16/2020 00:02] seed@ubuntu:~/Desktop$ cat file3
asdqwezx0
[06/16/2020 00:02] seed@ubuntu:~/Desktop$ 
```

如图两个文件的md5值。

写一个小程序计算有多少位是相同的



```
D:\Project\Python\untitled4\venv\Scripts\python.exe D:/Project/Python/untitled4/set.py
0

Process finished with exit code 0
```

```python
def test():
    str1 = "6bae172a0054e9159e0fb2d9ccd4f3cf"
    str2 = "d37b9a914fa3d7f9d9e6ea2ad05a77ec"
    num = 0
    for i in range(0, len(str1)):
        if str1[i] == str2[i]:
            num = num + 1
            print(i, str1[i])

    print(num)
```

Task 4: One-Way Property versus Collision-Free Property

单向性

开头首24bit对应的字母	5位char	查询字数
123	0101c7e810	12933871
133	010247f267	21194683
124	101720695	7349249
126	0101fd280b	16396256
127	0101942f92	9570551
129	0101a2bc8d	10516851
130	01017999fe	7842014
平均值		12257639.29

其中125、128这两次的程序由于程序的未知错误，没有给出结果。

无碰撞性

加密文本	5位char	查询字数
Test Message	01013b1f46	3779170
AAAAAAAAAAAAAAAAa	0101c5dcb6	12800927
AAAbbAAAAAAAAAAAa	010177106b	7676882
AAAABBBBBAa	0101044f99	215118
23334wwdasd4444	101259160	2377716
12121asdassdasd4444	01017af367	7929838
0101213ee2	0101213ee2	2096581
平均值		5268033.143

其中加密文本234wwdasda 由于程序的未知错误，没有给出结果

结果

从上面的结果看，无碰撞性更好蛮力破解。

数学角度

1) 任意给定H(m)属于(0,1)^32的空间内，找到m属于(0,1)^40，这是单向性。

2) m属于(0,1)^40，找到同一hash值，可以与前面的hash值比较，概率较高。

附录代码：

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

int compare24(unsigned char* str1, unsigned char* str2){
    for(int i=0; i<3; i++){
        if(str1[i] != str2[i]){
            return 0;
        }
    }
    return 1;
}

unsigned char* result0(){
    EVP_MD_CTX *mdctx;
    const EVP_MD *md;
    unsigned char mess1[] = "Test Message";
    unsigned char* md_value = (unsigned char *)(malloc(sizeof(unsigned char)*EVP_MAX_MD_SIZE));
    unsigned int md_len, i;


    md = EVP_get_digestbyname("md5");


    mdctx = EVP_MD_CTX_new();
    EVP_DigestInit_ex(mdctx, md, NULL);
    EVP_DigestUpdate(mdctx, mess1, strlen(mess1));
    EVP_DigestFinal_ex(mdctx, md_value, &md_len);
    EVP_MD_CTX_free(mdctx);
    return md_value;
}

unsigned char* crack0(char* mess1){
    EVP_MD_CTX *mdctx;
    const EVP_MD *md;
    unsigned char* md_value = (unsigned char *)(malloc(sizeof(unsigned char)*EVP_MAX_MD_SIZE));
    unsigned int md_len, i;
    md = EVP_get_digestbyname("md5");
    mdctx = EVP_MD_CTX_new();
    EVP_DigestInit_ex(mdctx, md, NULL);
    EVP_DigestUpdate(mdctx, mess1, strlen(mess1));
    EVP_DigestFinal_ex(mdctx, md_value, &md_len);
    EVP_MD_CTX_free(mdctx);
    return md_value;
}

int crack1(){
    //0
    uint64_t flag=0;

    // 1
    unsigned char * temp0 = crack0("121212wwwwwwwasdassdasd4444");

    // 2
    unsigned char* text = (char *)malloc(sizeof(char)*1024);
    memset(text, 1, 1023);
    text[5] = '\0';
    for(int i0=1; i0<256; i0++){
        text[0] = i0;
        for(int i1=1; i1<256; i1++){
            text[1] = i1;
            for(int i2=1; i2<256; i2++){
                text[2] = i2;
                for(int i3=1; i3<256; i3++){
                    text[3] = i3;
                    for(int i4=1; i4<256; i4++){
                        text[4] = i4;
                        flag++;
                        unsigned char * temp1 = crack0(text);
                        if(compare24(temp1, temp0)==1){
                            for (int i = 0; i < 5; i++)
                                printf("%02x", text[i]);
                            printf("\n");
                            printf("%lld",flag);
                            return 0;
                        }
                    }
                }
            }
        }

    }
    free(temp0);
    free(text);
}

int crack2(){
    // 0
    uint64_t flag=0;

    // 1
    unsigned char * temp0 = "13045";

    // 2
    unsigned char* text = (char *)malloc(sizeof(char)*1024);
    memset(text, 1, 1023);
    text[5] = '\0';
    for(int i0=1; i0<256; i0++){
        text[0] = i0;
        for(int i1=1; i1<256; i1++){
            text[1] = i1;
            for(int i2=1; i2<256; i2++){
                text[2] = i2;
                for(int i3=1; i3<256; i3++){
                    text[3] = i3;
                    for(int i4=1; i4<256; i4++){
                        text[4] = i4;
                        flag++;
                        unsigned char * temp1 = crack0(text);
                        if(compare24(temp1, temp0)==1){
                            for (int i = 0; i < 5; i++)
                                printf("%02x", text[i]);
                            printf("\n");
                            printf("%lld",flag);
                            return 0;
                        }
                    }
                }
            }
        }

    }
    free(temp0);
    free(text);
}




int main(int argc, char *argv[])
{
    crack1();
    //crack2();
    exit(0);
}
```