ENCS5121
Information Security and
Computer Networks Laboratory

EXPERIMENT #4
Hash Length Extension Attack

Slides by: Mohamad Balawi
Updated By: Tariq Odeh

BIRZEIT UNIVERSITY

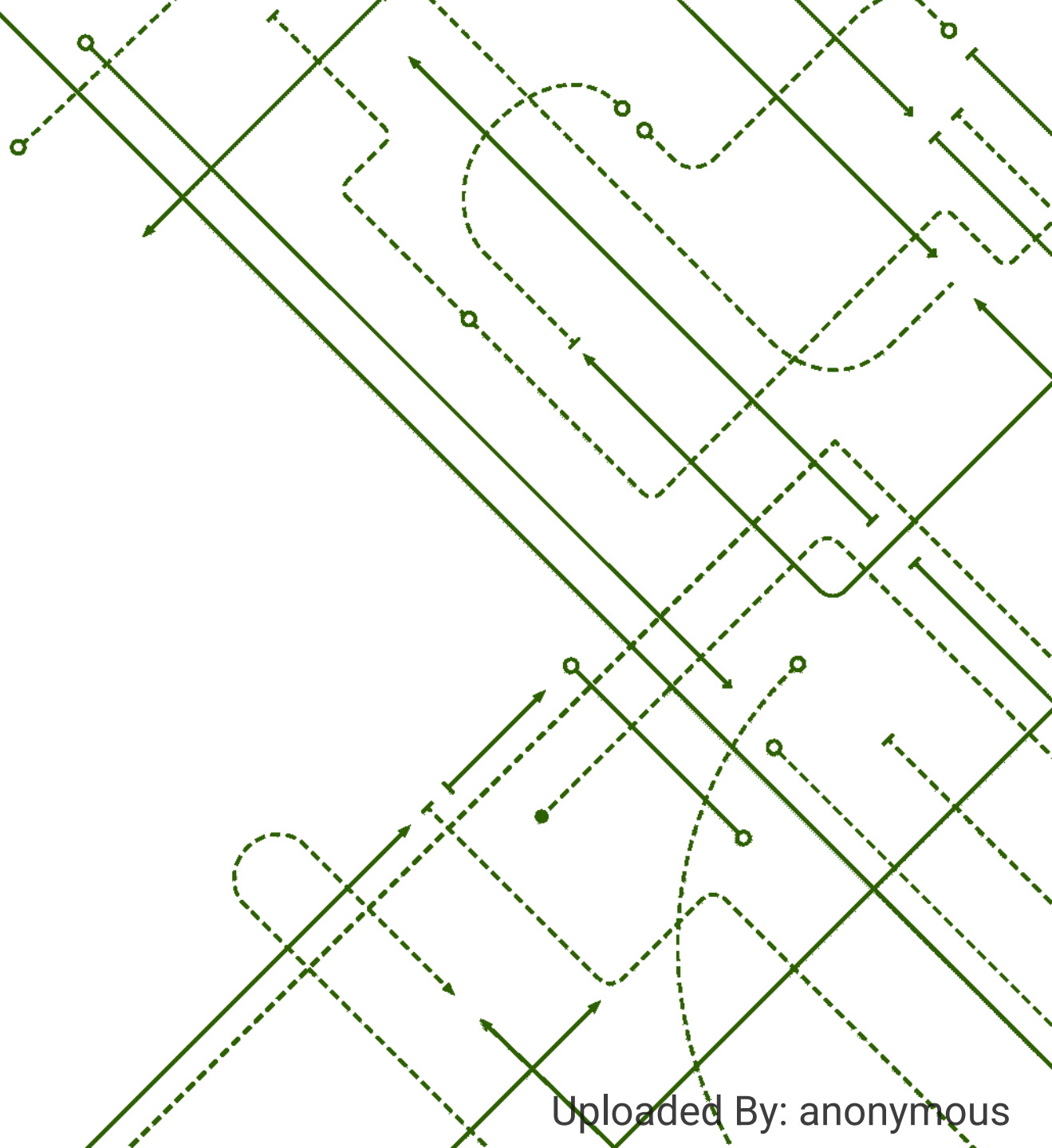
Overview

- **Objective:**
  - Understand and demonstrate a length extension attack on a Message Authentication Code (MAC).
- **Scenario:**
  - Client-server communication is vulnerable to MITM attacks, where an attacker can intercept, modify, and re-send requests. Servers need to verify request integrity to prevent tampering.
- **Task:**
  - Perform a length extension attack on a server, forge a valid command, and trick the server into executing the modified request.
- **Topics Covered:**
  - MAC calculation and verification.
  - Length extension attack mechanics.
  - Secure communication practices.

Outline

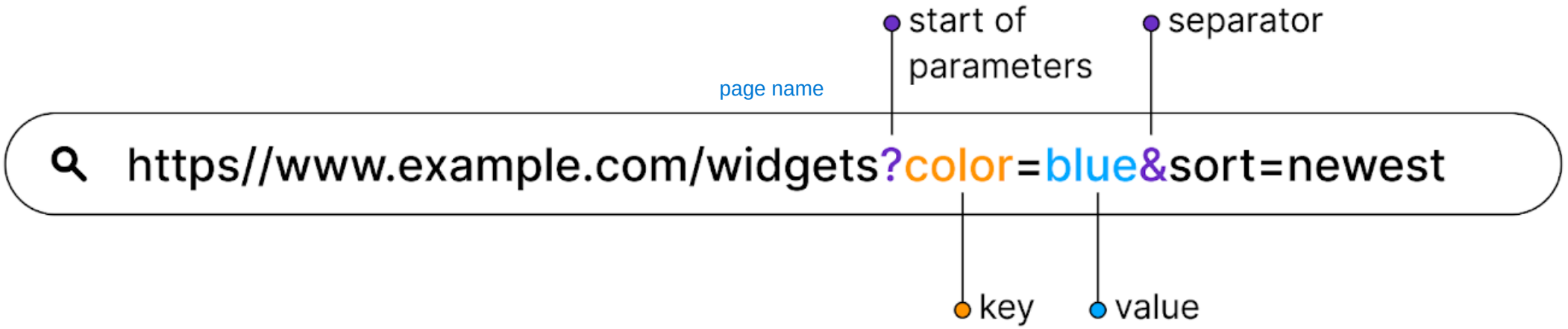
- **Introduction**
  - Web
  - MAC/HITM
  - Hashing/HMAC
- **Lab Environment**
- **Task 1: Send Request to List Files**
- **Task 2: Create Padding**
- **Task 3: The Length Extension Attack**
- **Task 4: Attack Mitigation using HMAC**

Introduction



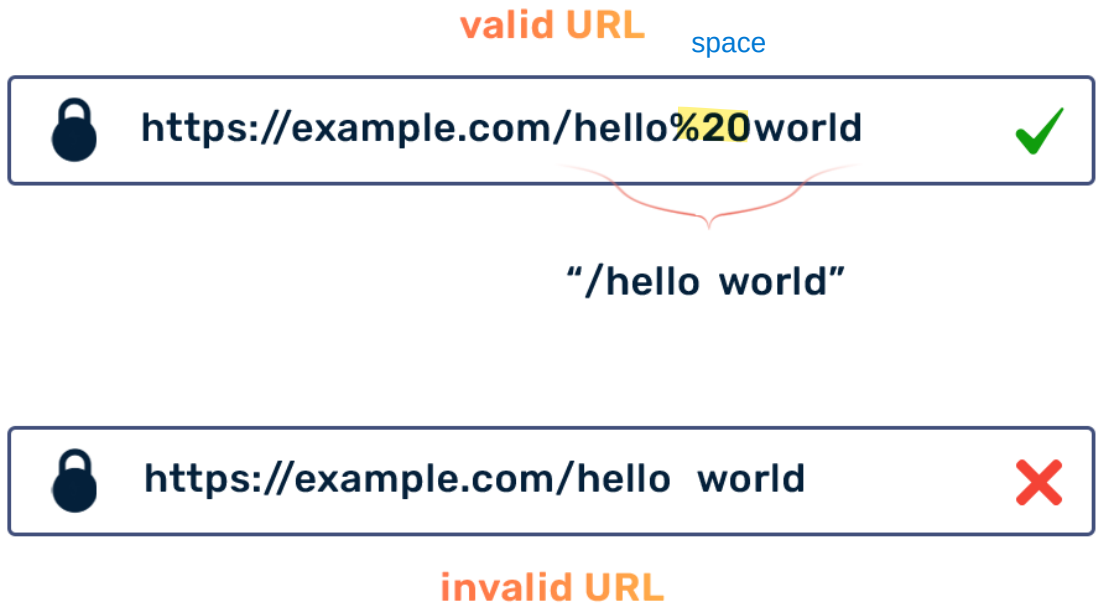
Uniform Resource Locator (URL) Parameters

- **URL parameters** (also known as query strings or URL query parameters) are elements inserted in your URLs to help you filter and organize content.



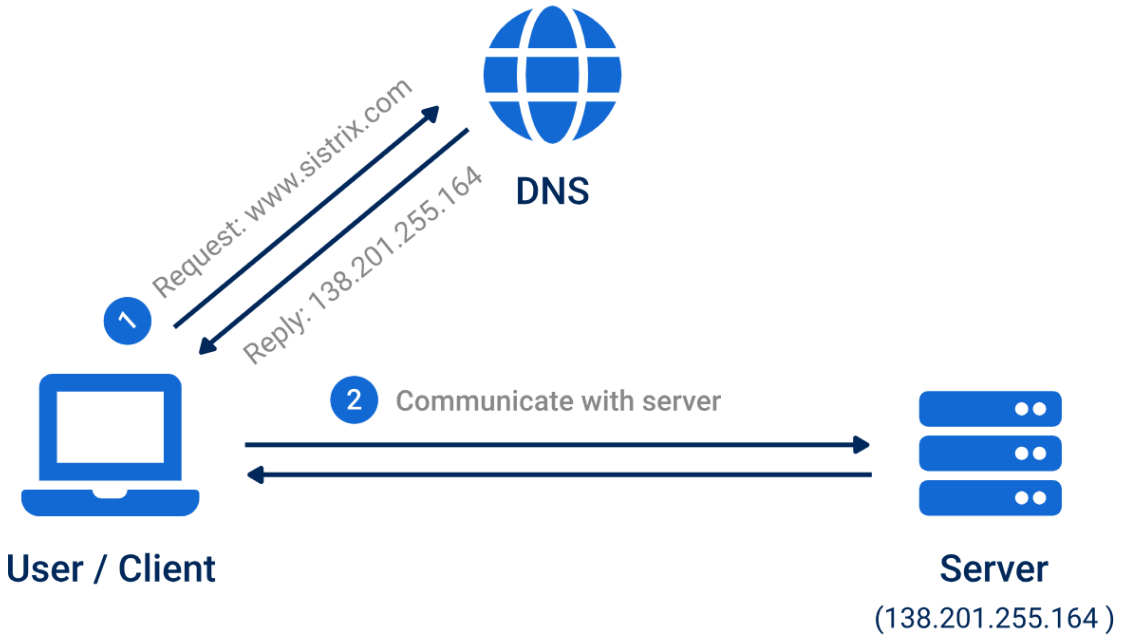
Uniform Resource Locator (URL) Encoding

- **URL encoding (also known as Percent encoding)**
  - It is used to convert characters that are not universally accepted by URLs into a format that can be transmitted safely.
  - Additionally, certain reserved characters with special meanings in URLs, such as '&' and '?', are also encoded to avoid misinterpretation by web servers.
  - Ampersand ('&') is encoded as %26
  - Question mark ('?') is encoded as %3F



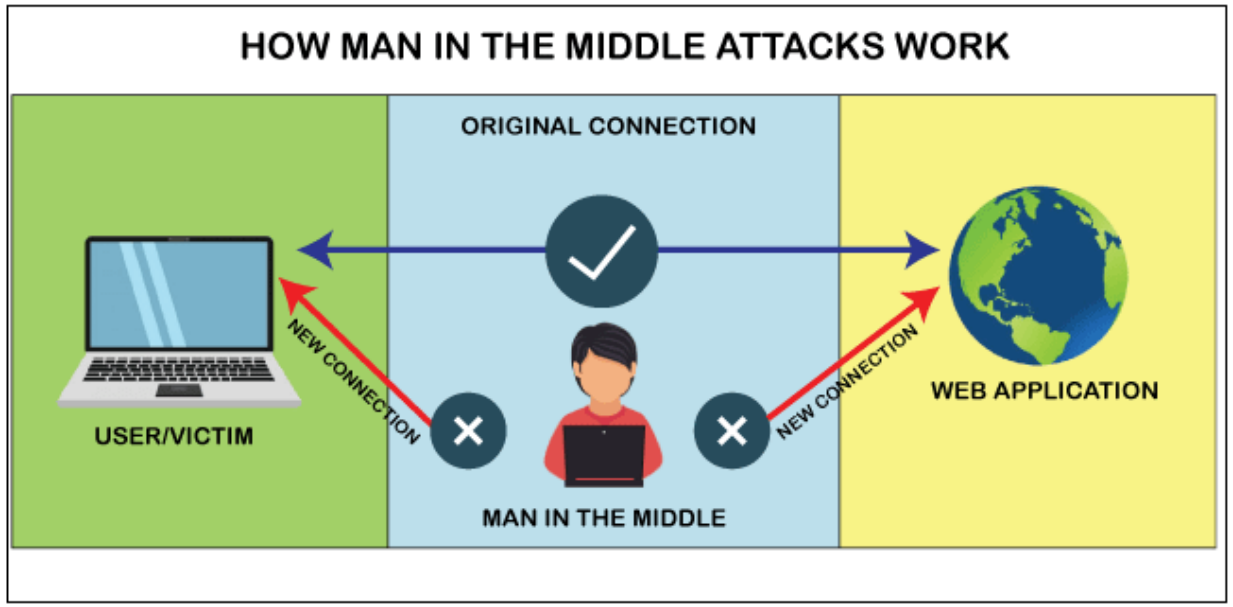
/etc/hosts file

- **/etc/hosts** is a text file on Unix-like operating systems (such as Linux and macOS) that maps hostnames to IP addresses. It is a local DNS (Domain Name System) resolver that allows the system to resolve hostnames to IP addresses without querying a DNS server.



MITM (Man in the Middle) attacks

- When a client and a server communicate over the internet, they are subject to MITM attacks. An attacker can intercept the request from the client. The attacker may choose to modify the data and send the modified request to the server.



MAC (Message Authentication Code)

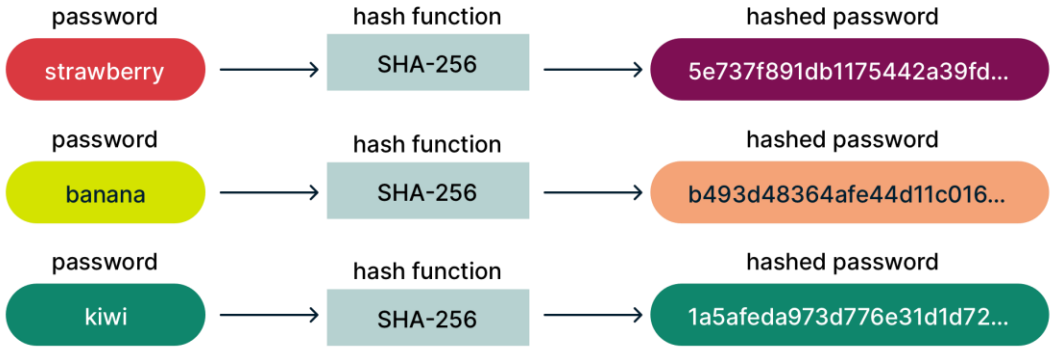
- In a MITM scenario, the server needs to verify the integrity of the request received. The standard way to verify the integrity of the request is to attach a tag called MAC to the request. There are many ways to calculate MAC, and some of the methods are not secure.

- MAC is calculated from a secret key and a message. A naive way to calculate MAC is to concatenate the key with the message and calculate the one-way hash of the resulting string. This method seems to be fine, but it is subject to an attack called length extension attack.

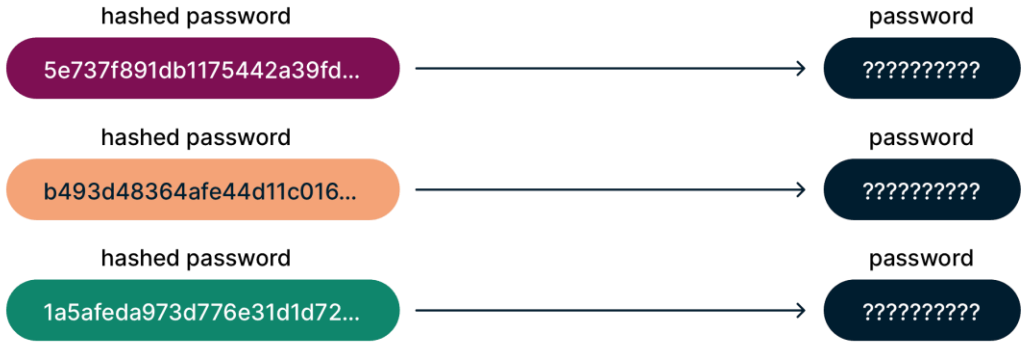
$$MAC = Hash(\text{Key} \,||\, \text{Msg})$$

Hashing usage example

- We use hashing functions to store passwords in a database, so even if the database gets leaked somehow, the original passwords remain unknown.



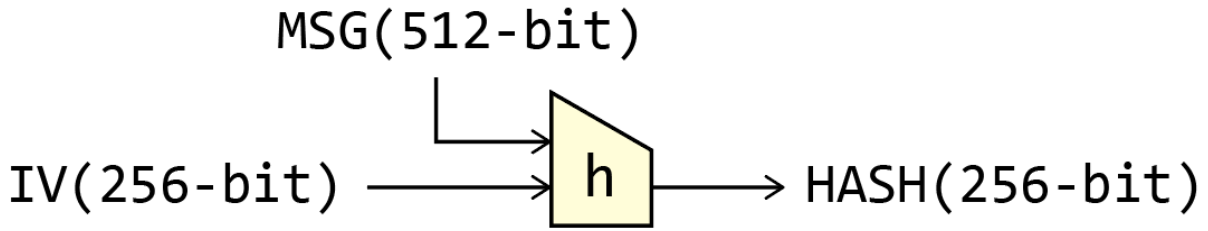
- During login attempts, user input is hashed and compared to the hashed password stored in the database for authentication.



SHA256

- SHA256 is a cryptographic hash function that generates a fixed-size output for any given input. padding for sure
  It operates on blocks of data. This algorithm is widely used in various security applications such as digital signatures, message authentication codes, and blockchain technology.

  - Based on Merkle–Damgard construction
  - Block size is 512-bit (64 bytes)
  - Output size is 256-bit (32 bytes) (hence the name SHA256)



SHA256 Padding

- The block size of SHA-256 is 64 bytes, so a message M will be padded to the multiple of 64 bytes during the hash calculation.
- Paddings for SHA256 consist of one byte of \x80, followed by a many 0's, followed by an 8 bytes length field (the length is the number of bits in the M).

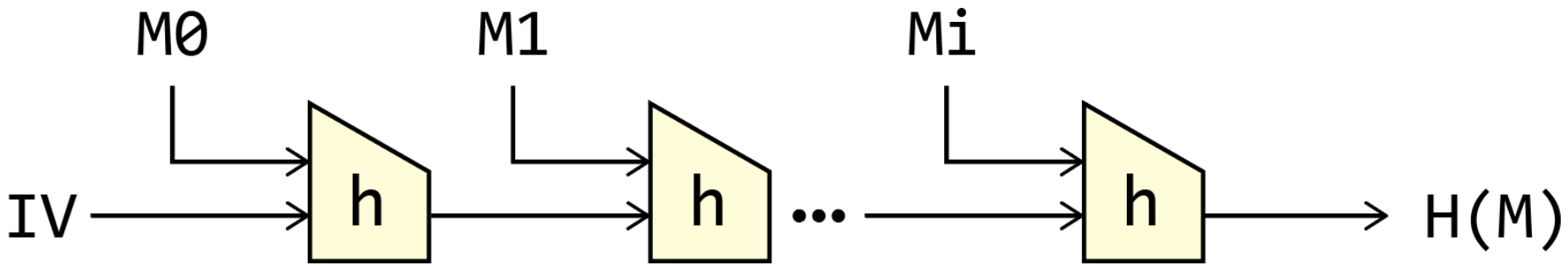
- **Example:** $\boldsymbol{M} = \text{“}\boldsymbol{This\ is\ a\ test\ message}\text{”}$
  - The length of M is 22 bytes (176-bit) (0xB0), so the padding is 64 - 22 = 42 bytes.
  - "This is a test message\x80\x00\x00\x00\x00\x00\x00\x00\x00\x00
    \x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
    \x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\xB0"

Merkle–Damgard construction

- In Markle-Damgard construction, the message is split into blocks and fed into a chain of functions, the output of the final block is the output of the whole construction.

- Given the following message: `M = M0 || M1 || ... || Mi`

- We can obtian its hash as follows:



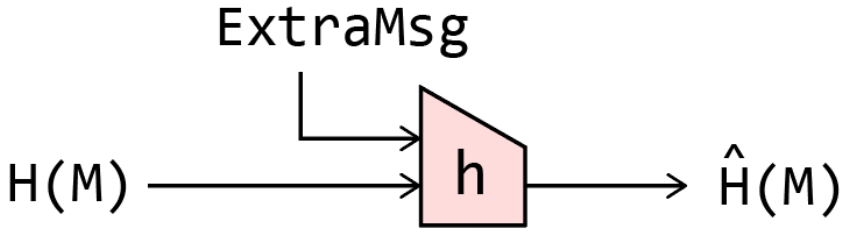
Length Extension Attack

We can take advantage of Merkle–Damgard construction to launch length extention attack:

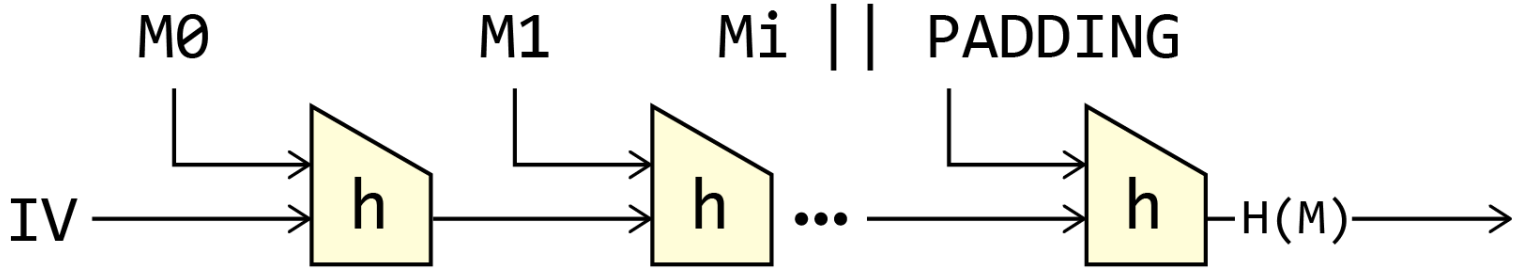
- Allows us to generate a valid MAC without knowing the secret key.
- How it works?
  - The attacker appends a new block to the message.

$$\hat{M} = M0 \,||\, M1 \,||\, ... \,||\, Mi \,||\, PADDING \,||\, EXTRA\_MSG$$

  - The attacker appends a new hashing stage.



Length Extension Attack



After that the attacker can send $\hat{M}$ message along with $\hat{H}(M)$ MAC.

Note that SHA256 does the padding on its own, so the attacker doesn't have to worry about it.

HMAC (Hash-based Message Authentication Code)

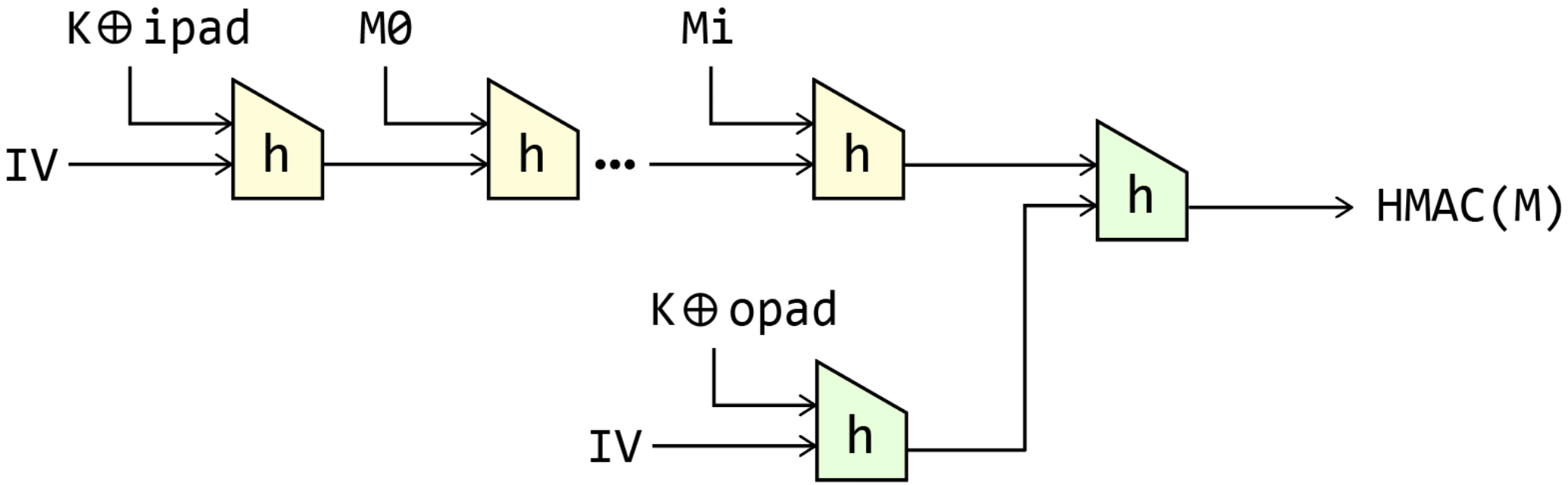
- HMAC is resistant to length extension attacks.

attacker messg.

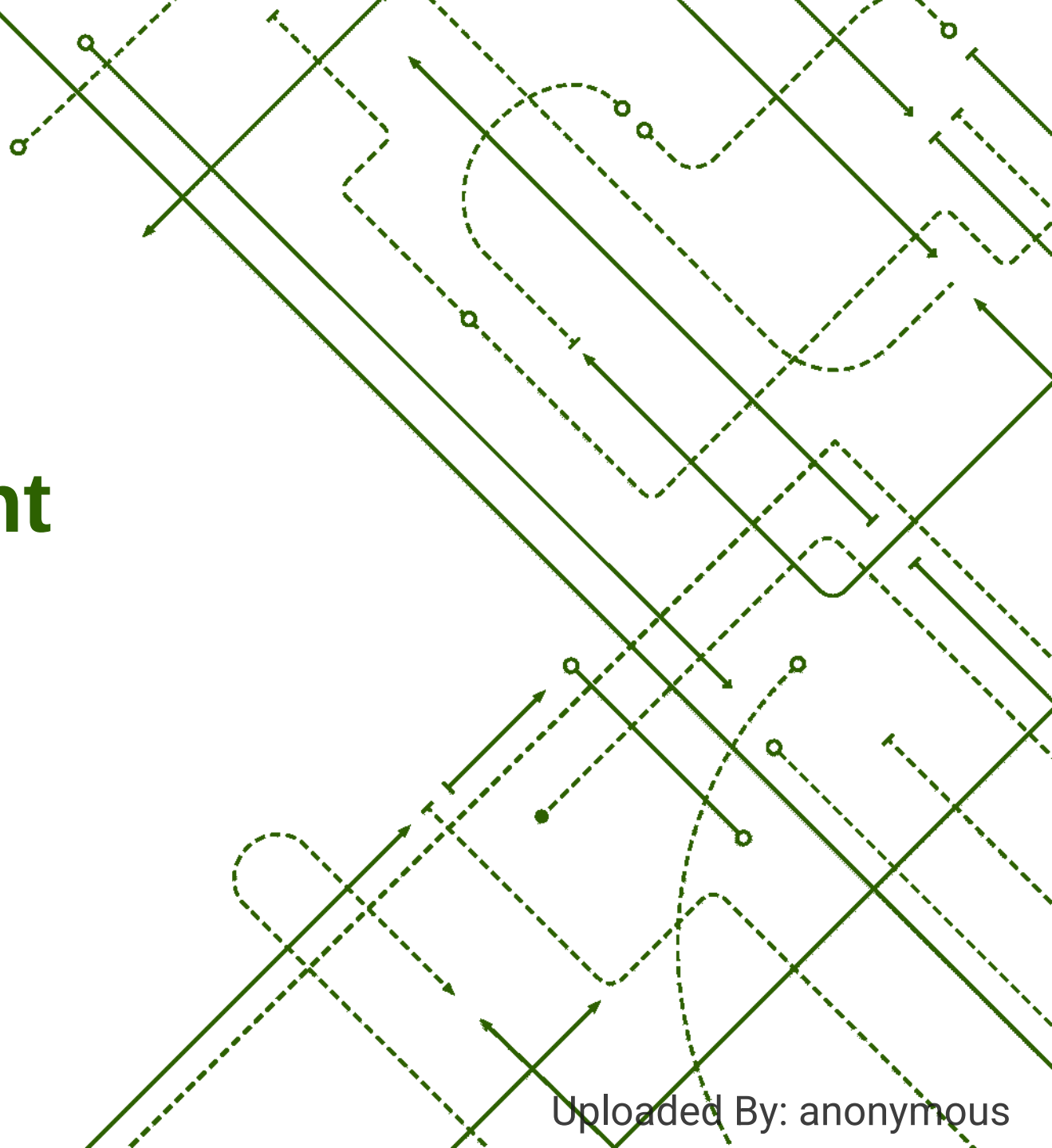
$$\mathrm{HMAC}(K, m) = \mathrm{H}\left((K \oplus opad) \parallel \mathrm{H}\left((K \oplus ipad) \parallel m \right) \right)$$

- It prevents the attack by re-hashing the result.
- It adds two hashing blocks to the our naive approach.

HMAC Diagram



Lab Environment



Lab setup

1. **Download Labsetup.zip to your VM and Unzip, then enter the Labsetup folder.**

   https://seedsecuritylabs.org/Labs_20.04/Crypto/Crypto_Hash_Length_Ext/

2. **Open the /etc/hosts file:**

```
sudo nano /etc/hosts
```

3. **Add the following IP-to-Hostname mapping to the end of /etc/hosts file:**

```
10.9.0.80    www.seedlab-hashlen.com
```

4. **Starting the server by navigating to labsetup (where docker-compose.yml is located), then executing the following commands:**

```
dcbuild
dcup
```

Server

The server code is in the Labsetup/image_flask/app folder. It has two directories:

www directory contains the server code.

LabHome directory contains a secret file, and the key used for computing the MAC.

The server program accepts the following commands:

```
lstcmd        # list all the files in the LabHome folder.
download      # print the contents of a specified file from LabHome directory
```

How to contact the server?

We will use Mozilla Firefox browser to contact the server.

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

Of course we need to replace the following values:

`<name>`	Put yourname.
`<your_uid>`	Put your chosen UID.
`lstcmd=1`	Put your commands.
`<mac>`	Put your MAC after calculating.

How to calculate MAC in this experiment?

$$MAC = Hash(\text{Key} \,||\, \text{Msg})$$

$$MAC = sha256(\text{Key} \,||\, \text{Msg})$$

$$MAC = sha256(\text{Key} \,||\, \text{URL\_PARAMETERS})$$

$$MAC = sha256(\text{Key} : \text{URL\_PARAMETERS})$$

Example:

$$MAC = sha256(123456{:}myname{=}JohnDoe\&uid{=}1001\&lstcmd{=}1)$$

Calculate SHA256 using a shell command

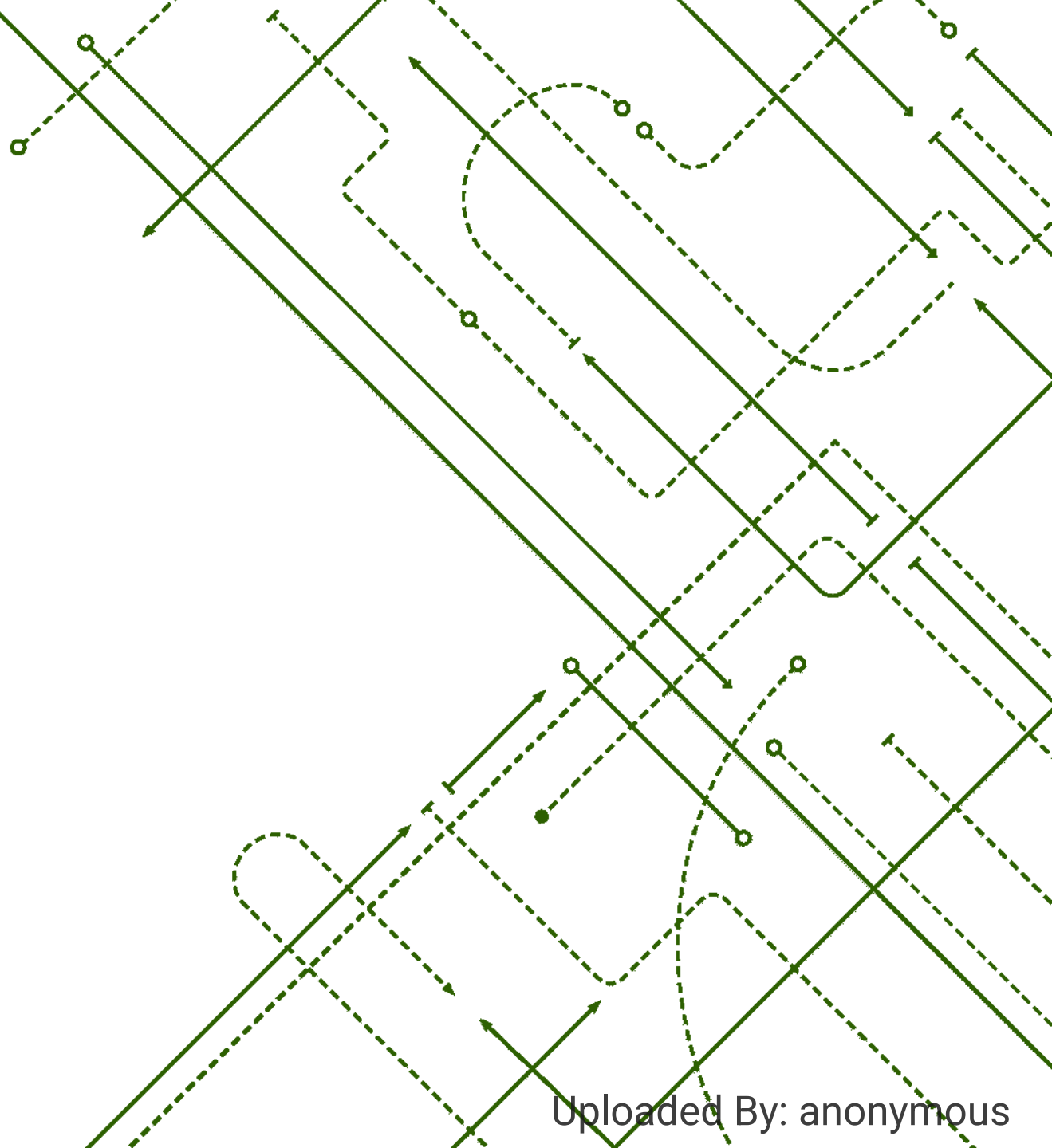
- **We can use `sha256sum` command.**

mac add.

```
echo -n "123456:myname=JohnDoe&uid=1001&lstcmd=1" | sha256sum

7d5f750f8b3203bd963d75217c980d139df5d0e50d19d6dfdb8a7de1f8520ce3
```

Tasks



Task 1: Send Request to List Files

- You should put your full name in the url parameters using **camelCase** (mohamadBalawi).
- The value of the **UID** and the **Key** should be obtained from the LabHome/key.txt file
- Send a download command to server and and show that you can get the results back.

Task 2: Create Padding

- You should put your full name in the url parameters using camelCase (mohamadBalawi).
- The value of the UID and the Key should be obtained from the LabHome/key.txt file
- You need construct the padding for the following message:
  - **<key>:myname=<name>&uid=<uid>&lstcmd=1**

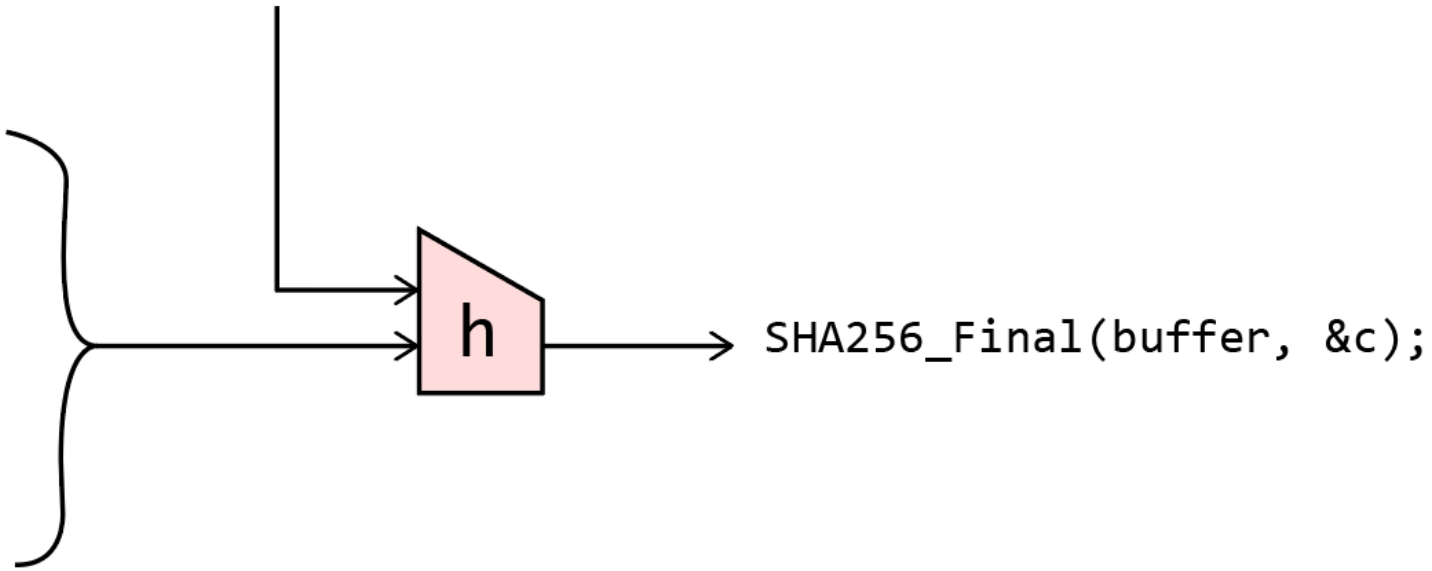
Task 3: The Length Extension Attack

- generate a valid MAC for the following request:
  - **<key>:myname=<name>&uid=<uid>&lstcmd=1**
- Use length_ext.c program to generate a new MAC for the following extended request:
  - **myname=<name>&uid=<uid>&lstcmd=1<padding>&download=secret.txt**
- You can get <padding> from Task2.
- Send the constructed request to the server and show that you can successfully get secret.txt.

Task 3: The Length Extension Attack (Cont.)

```
SHA256_Update(&c, "Extra message", 13);
```

```
c.h[0] = htole32(0x6f343800);
c.h[1] = htole32(0x1129a90c);
c.h[2] = htole32(0x5b163792);
c.h[3] = htole32(0x8bf38bf2);
c.h[4] = htole32(0x6e39e57c);
c.h[5] = htole32(0x6e951100);
c.h[6] = htole32(0x5682048b);
c.h[7] = htole32(0xedbef906);
```



Task 4: Attack Mitigation using HMAC

- Modify **`verify_mac`** function in **`Labsetup/image_flask/app/www/lab.py`** to use HMAC instead of our naive approach.

```
- real_mac = hmac.new(bytearray(key.encode('utf-
  8')),msg=message.encode('utf-8',
  'surrogateescape'),digestmod=hashlib.sha256).hexdigest()
```

- Rebuild and restart the containers (**`dcbuild`** then **`dcup`**).
- Repeat the same length extention attack that we did in Task 3.