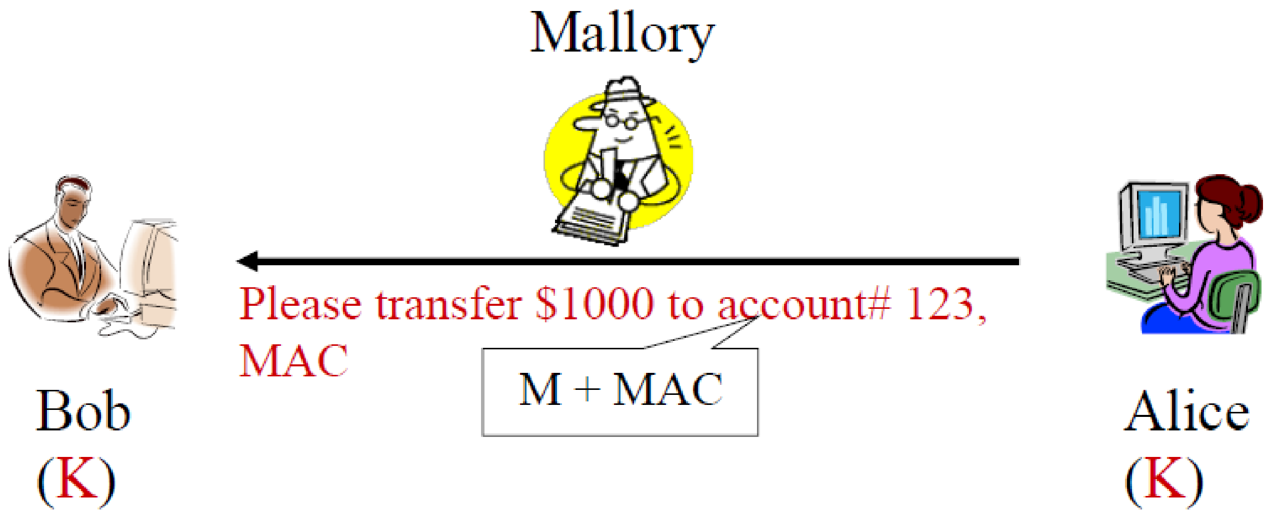
Message Authentication Code - MAC

- Goal: to ensure data integrity
- Message + Key -> Hashing Algorithm -> MAC
- A keyed message digest
- Often used for authenticating data sent over an insecure network
- At the sending:
  - Compute the MAC of the outgoing message using a key
  - Send over the message, and the MAC.
- At the receiving end:
  - Received the incoming message and the MAC
  - The same key is used to produce MAC' based on the received message
  - The MAC' will be compared to MAC
  - This will determine whether the message has been tampered.

Why MAC ?



- On top of to prevent man-in-the-middle attack similar to the keyless message digest.
- MAC allows the recipient of the message to authenticate that the message's sender has the shared secret key.
- Integrity + Authenticity checking

The building blocks –

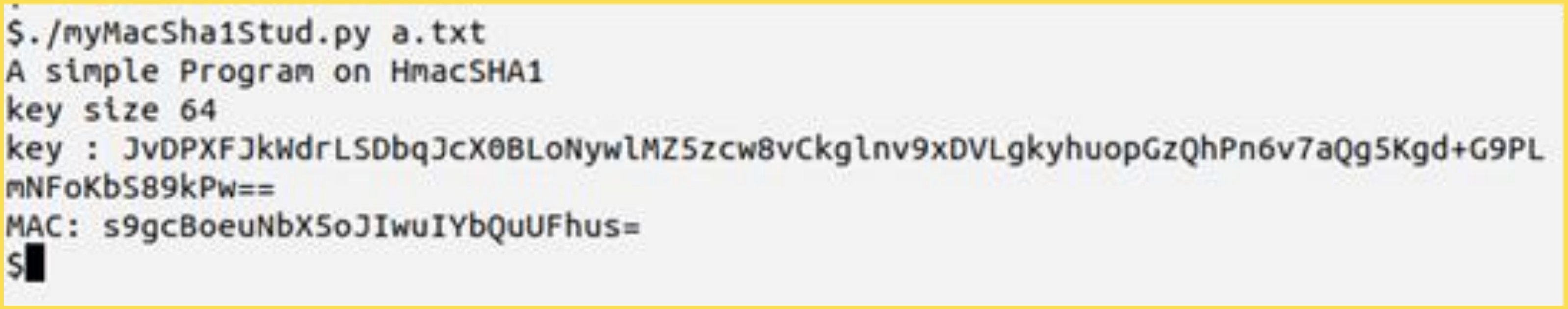
- For Hash-based message authentication code (HMAC)
- import hmac
- from Cryptodome.Random import get_random_bytes
- Syntax for object instantiation

  key = get_random_bytes(32)

  hmacobject = hmac.new(<key>, digest=<name of the hash function>)
- Example
  - hma1 = hmac.new(key,digestmod="md5")
  - hma2= hmac.new(key,digestmod="sha1")
- Usage
  - hma1.update(message)
    - message is a bytes-like object
    - Repeated calls are equivalent to a single call with the concatenation of all the arguments.
  - hma1.digest()
    - Returns message digest in bytes
  - hma1.hexdigest()
    - Returns message hexdigest in string type (Representation in hexadecimal digits)
- Reference – for more information
  - https://docs.python.org/3/library/hmac.html

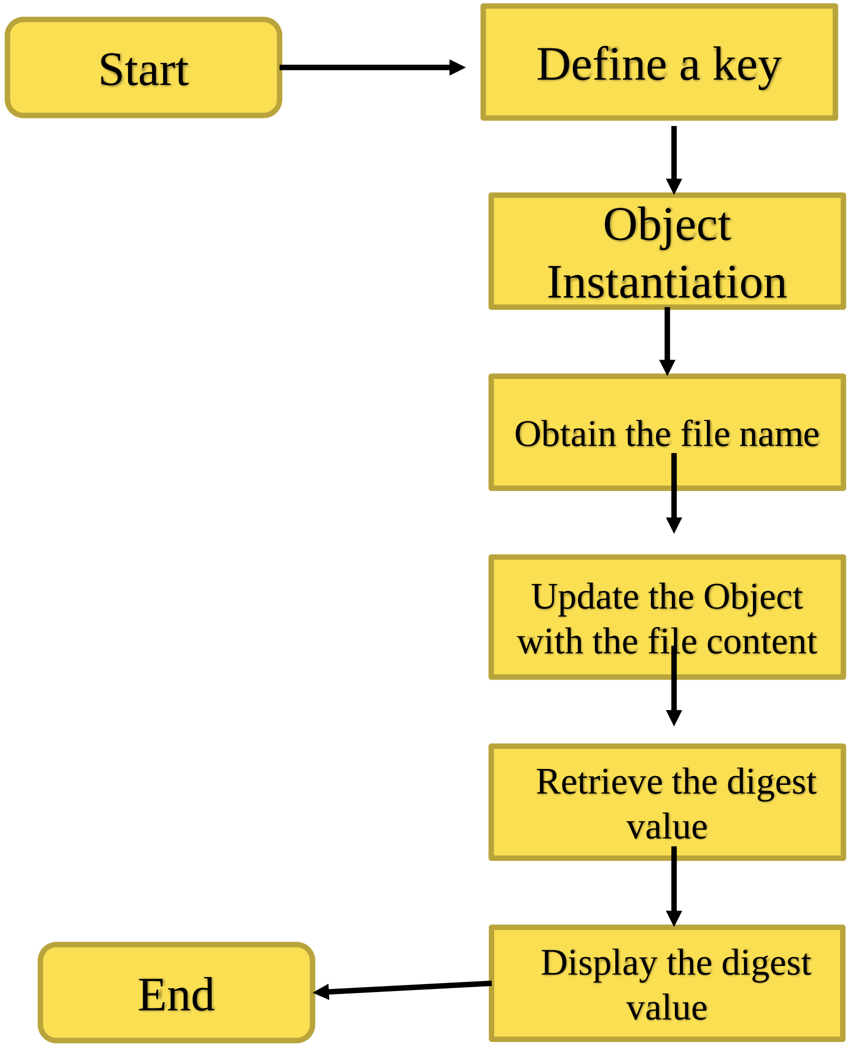
Demo: The Hash Mac

- Sample output:



```
$./myMacSha1Stud.py a.txt
A simple Program on HmacSHA1
key size 64
key : JvDPXFJkWdrLSDbqJcX0BLoNywlMZ5zcw8vCkglnv9xDVLgkyhuopGzQhPn6v7aQg5Kgd+G9PL
mNFoKbS89kPw==
MAC: s9gcBoeuNbX5oJIwuIYbQuUFhus=
$
```

Flow Chart – Hashed MAC Digest



```
key = "abckey".encode()

h = hmac.new(key,digestmod="sha1")

fn = sys.argv[1]

with open(fn) as f:
        text=f.read()
        hmac.update(text.encode())

d = hmac.digest()
b64=base64.b64encode(d)

print(b64.decode())
```

Template of myMacSha1Stud.py

```
#!/usr/bin/env python3
#ST2504 - ACG Practical - myMacSha1Stud_skel.py
from Cryptodome.Random import get_random_bytes
import hmac, base64
import sys
# main program starts here
argc = len(sys.argv)
if argc != 2:
    print("Usage : {0} <file name>".format(sys.argv[0]))
    exit(-1)
try:
    with open(sys.argv[1]) as f:
        content=f.read()    # read in the entire text file
        print("A simple Program on HmacSHA1")
        keysize=hmac.HMAC.blocksize # retrieve the default block size
        print("key size {0}".format(keysize))
        # insert your code here to generate a random key

        # display the key in base64 encoded bytes in UTF8 format
        print("key : {0}".format(base64.b64encode(key).decode()))
        # insert your code here to instantiate a sha1 hmac object, hma .

        # insert your code here to use hma to compute the hmac of content.

        # insert your code here to display the HMAC digest in
        # base64 encoded bytes in UTF8 format

except:
    print("Invalid file argument!")
```