

CSCI 476: Computer Security

Hashing

Reese Pearsall
Fall 2022

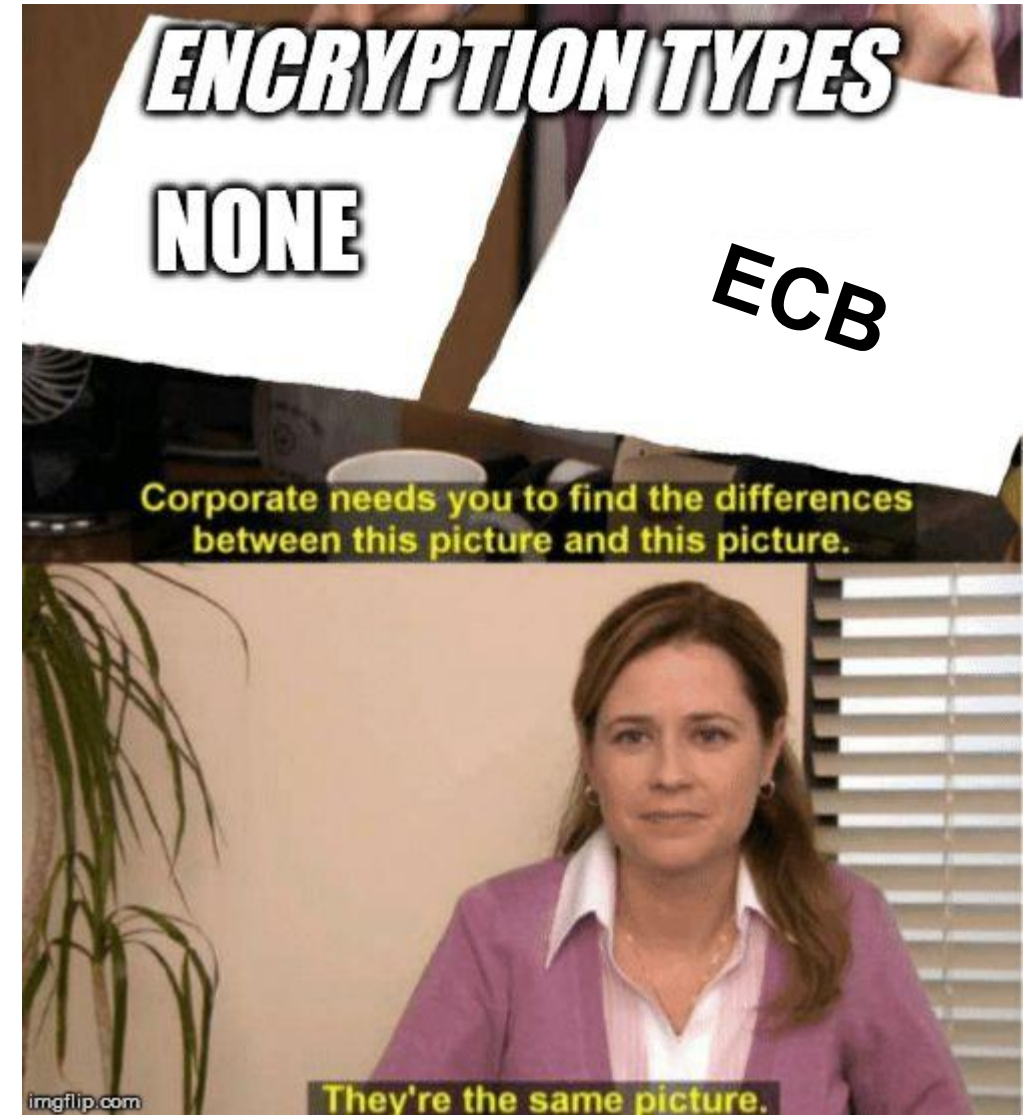
Announcement

Lab 8 (Secret-Key Encryption)
due Sunday 20th

Rest of semester dates are
posted

Extra credit will be applied at
end of semester

How to corrupt a ciphertext

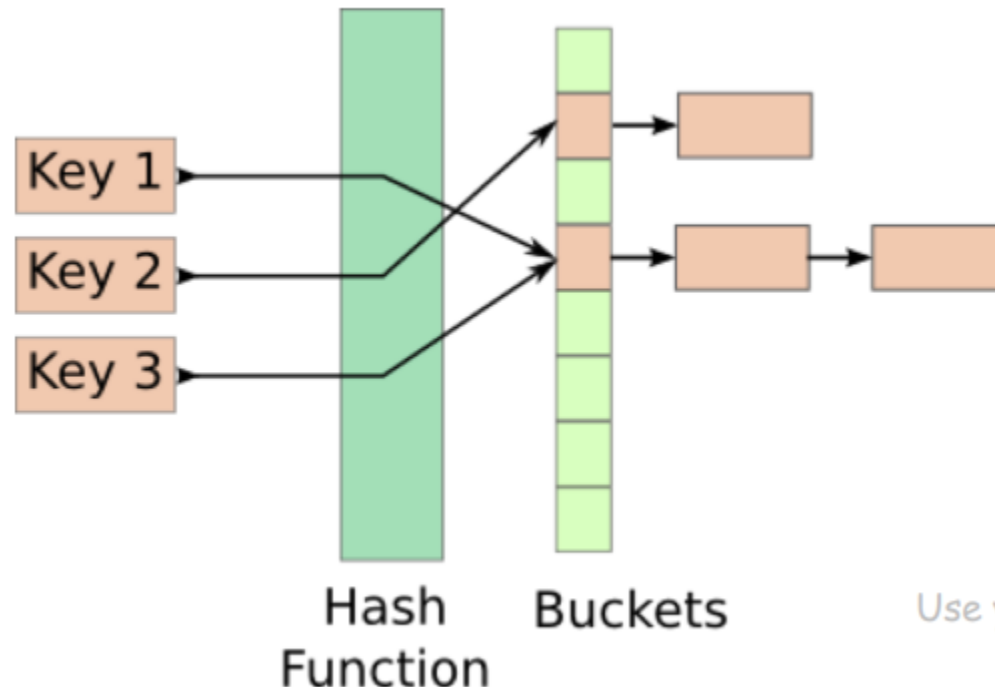


Hash Functions

Hash Functions map arbitrary size data to data of fixed size

- An essential building block in cryptography, with desirable practical and security properties

Ex. $f(x) = x \bmod 100$



How many buckets?

What to do if two keys map to the same bucket?

Collisions happen...

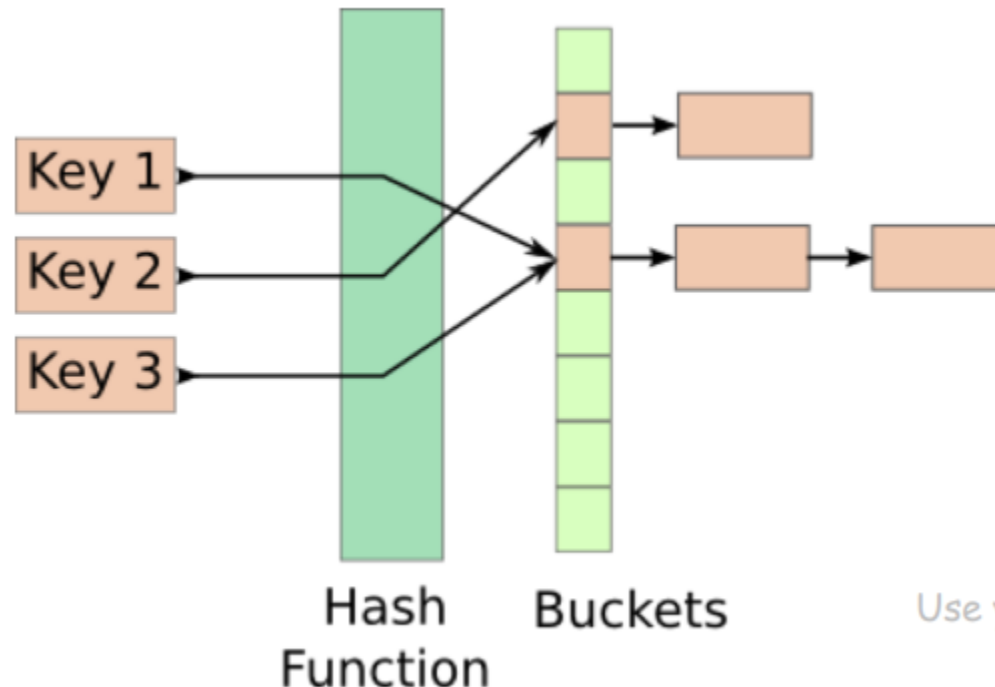
Use your favorite collision resolution technique
(open addressing, chaining, etc.)

Hash Functions

Hash Functions map arbitrary size data to data of fixed size

- An essential building block in cryptography, with desirable practical and security properties

Ex. $f(x) = x \bmod 100$



How many buckets?

What to do if two keys map to the same bucket?

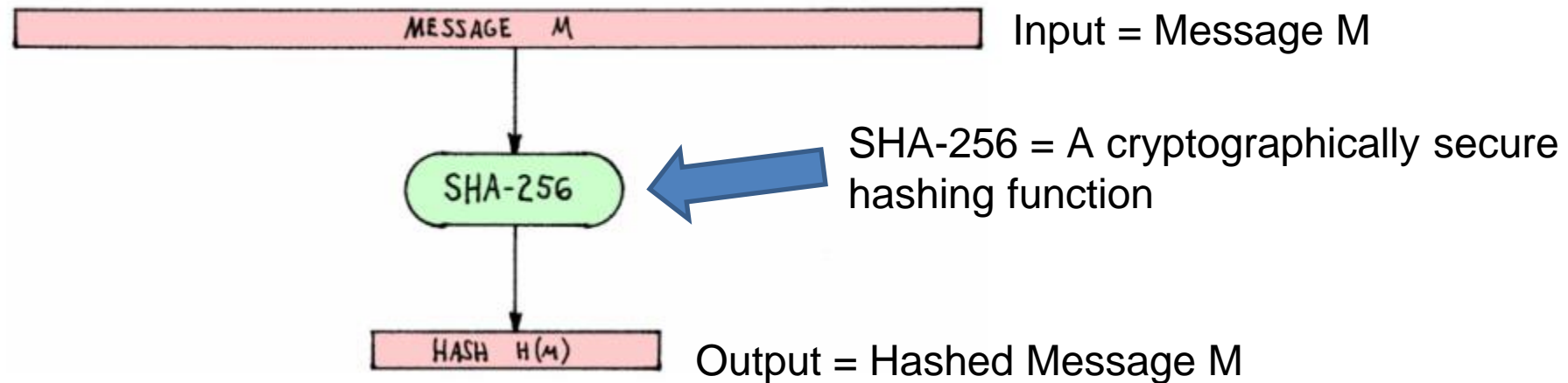
Collisions happen...

Use your favorite collision resolution technique
(open addressing, chaining, etc.)

Hash Functions

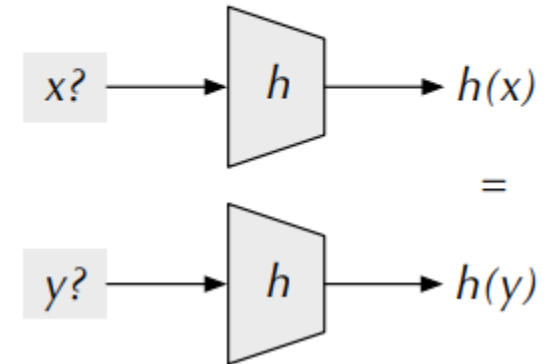
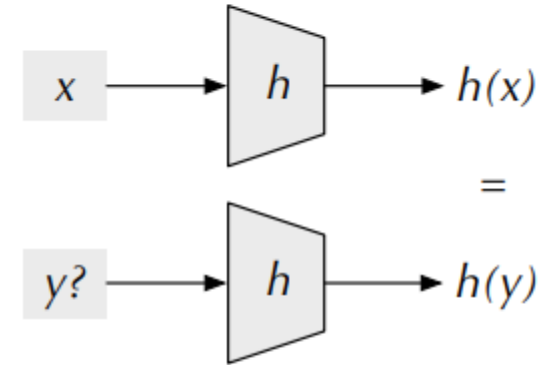
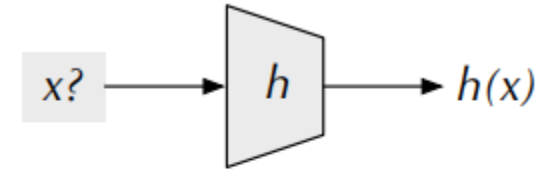
Cryptographic Hash Functions map arbitrary size data to data of fixed size

- But with **three** additional important properties



Hash Functions Properties

- **Preimage Resistance ("One-Way")**
Given $h(x) = z$, hard to find x
(or any input that hashes to z for that matter)
- **Second Preimage Resistance**
Given x and $h(x)$, hard to find y s.t. $h(x) = h(y)$
- **Collision Resistance (or, ideally, "Collision Free")**
Difficult to find x and y s.t. $hash(x) = hash(y)$



Hash Functions Properties (tl;dr)

```
[11/15/22] seed@VM: ~$ md5sum copy.bmp  
bb52593852da21b95a8ab8ce64ca7261  copy.bmp
```

Gives an arbitrary size input a fixed-size unique* hash identifier

Hash values are very difficult to **reverse**. They were designed to be one-way

The go-to way to reverse a hash is through brute force

Computing Hashes with OpenSSL

```
[11/15/22]seed@VM:~$ openssl dgst -list
```

Supported digests:

-blake2b512	-blake2s256	-md4
-md5	-md5-sha1	-mdc2
-ripemd	-ripemd160	-rmd160
-sha1	-sha224	-sha256
-sha3-224	-sha3-256	-sha3-384
-sha3-512	-sha384	-sha512
-sha512-224	-sha512-256	-shake128
-shake256	-sm3	-ssl3-md5
-ssl3-sha1	-whirlpool	

Calculating the Hash for a text file with SHA 256

```
[11/15/22]seed@VM:~$ openssl dgst -sha256 cipher2.txt
```

```
SHA256(cipher2.txt)= ca795bd6cbdee2c4cb8a23a512f08223ba498a7317070b914d49321a2a43d538
```

Property of Hashes: One small change in file → will drastically change hash (avalanche effect)

```
[11/15/22]seed@VM:~$ echo "hi123" > message.txt
```

```
[11/15/22]seed@VM:~$ openssl dgst -sha256 message.txt
```

```
SHA256(message.txt)= 41603550d2a90f7a722c6a45b6a497ee075b6f70f3ec869aded568383f839b25
```

```
[11/15/22]seed@VM:~$ echo "hi122" > message.txt
```

```
[11/15/22]seed@VM:~$ openssl dgst -sha256 message.txt
```

```
SHA256(message.txt)= 556c6dfd6ec82ac31267b26a906b9620f1df472193467321960a2f743ee01874
```


Families of Hash Function

- **Message Digest**
 - Developed by Ron Rivest
 - Produces 128-bit hashes
 - Includes MD2, MD4, MD5, and MD6
- **Status of Algorithms:**
 - MD2, MD4 - severely broken (obsolete)
 - MD5 - collision resistance property broken; one-way property not broken
 - Often used for file integrity checking
 - No longer recommended for use!
 - MD6 - developed in response to proposal by NIST
 - Not widely used...

We will be focusing on MD5, and breaking MD5 in our Lab ☺

Families of Hash Function

- Secure Hash Algorithm

- Published by NIST
- Includes SHA-0, SHA-1, SHA-2, and SHA-3

- Status of Algorithms:

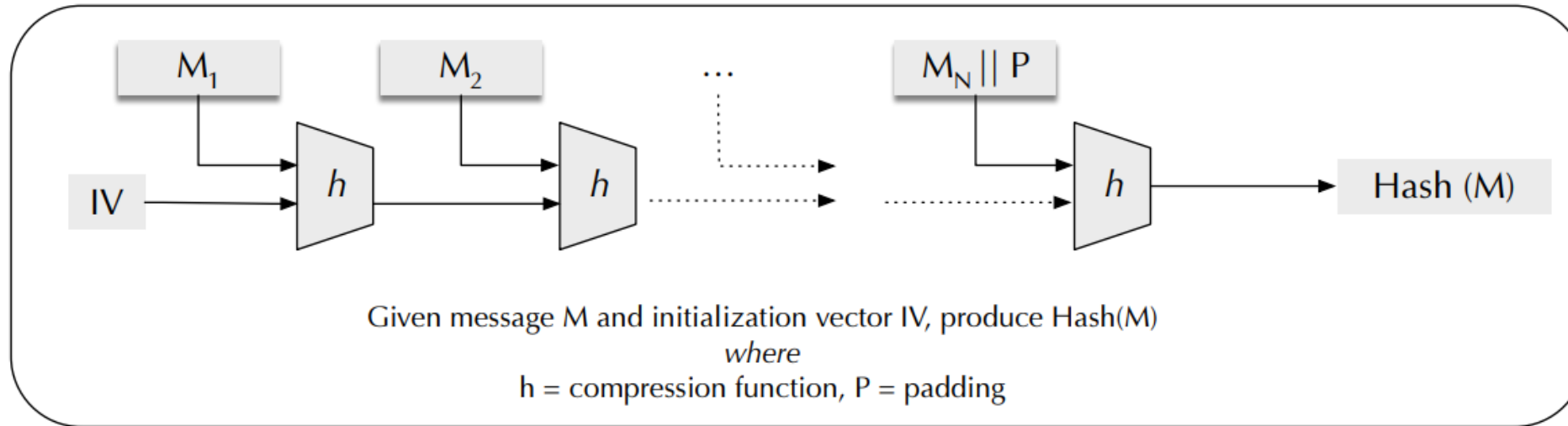
- SHA-0: withdrawn due to flaw
- SHA-1: Designed by NSA Collision attack found in 2017
- SHA-2: Designed by NSA
 - Includes SHA-256 and SHA-512 + other truncated versions;
 - No significant attack found yet...
- SHA-3: Not Designed by NSA
 - Released in 2015; not a replacement to SHA-2, but meant to be a genuine alternative
 - Has different construction structure ("Sponge Function") as compared to SHA-1 and SHA-2



<https://shattered.it>

How does MD5 work?

Most hash algorithms (e.g., MD5, SHA-1, SHA-2) use a Merkle-Damgård construction:



Davies-Meyer compression function uses a block cipher to construct a compression function
(e.g., SHA family uses this compression function)
Others are possible too...

```
[11/15/22]seed@VM:~$ echo "SADFLJKHASFLKSDJGFLAKDSJHASLFLKJHASDFLKJDSHAFISLDAUHFAILFGHASLK
DJGFHDSLKVJHSADLVKJNDSAVLKJSDAVLKDSJHGVDLSKJHGSALIGHUREIGUHOERAGIOUHASGKJASDHGSDLKJGFHASD
IGUHERIGUHAEGKLJHDSGKLDSJGHAOGIUHAERGIAUEPHGLAKJDSGHADSLKJGHDSAGIUHGAERLIGUHARES" > wut.
txt
[11/15/22]seed@VM:~$ openssl dgst -md5 wut.txt
MD5(wut.txt)= db806ca9d93fdc8bc4a6b76bd7e6432d
```

The **compression** of data is also a helpful application of hash functions

Calculating Hashes in Programming Languages

```
# Python 3 code to demonstrate the
# working of MD5 (string - hexadecimal)

import hashlib

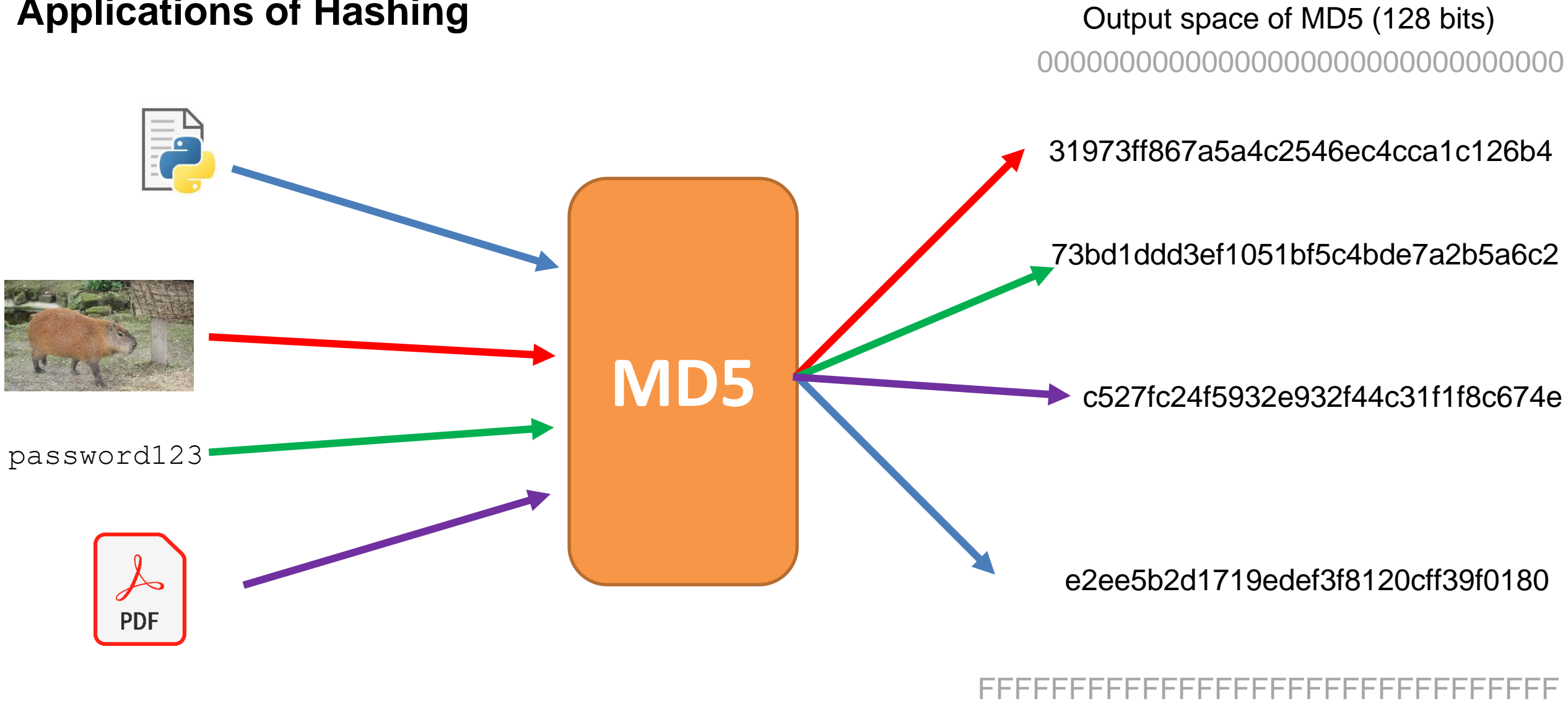
# initializing string
str2hash = "csci476"

# encoding csci476 using encode()
# then sending to md5()
result = hashlib.md5(str2hash.encode())

# printing the equivalent hexadecimal value.
print("The hexadecimal equivalent of hash is : ", end = "")
print(result.hexdigest())
```

Pretty much every
programming language
can calculate hashes

Applications of Hashing



What are some uses for hashing?

Applications of Hashing

Integrity Verification



`hello_world`

A CSCI 112 Student

Applications of Hashing

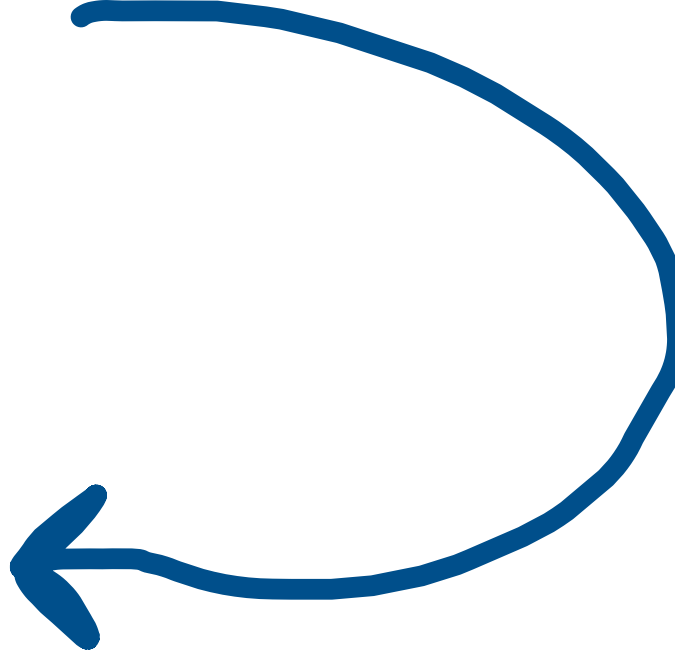
Integrity Verification



A CSCI 112 Student



hello_world



Sent to professor for grading



Instructor



hello_world

Applications of Hashing

Integrity Verification



A CSCI 112 Student



hello_world



Instructor



hello_world

```
#include <unistd.h>

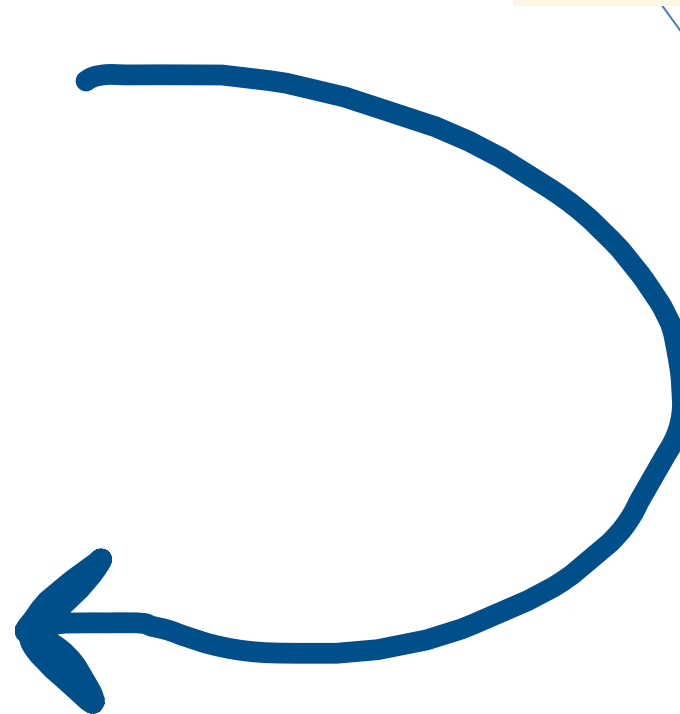
int main(void) {
    for (;;) {
        fork();
    }
}
```



hello_world



What if the message
got tampered with?



Applications of Hashing

Integrity Verification



A CSCI 112 Student



hello_world



Instructor



hello_world

```
#include <unistd.h>

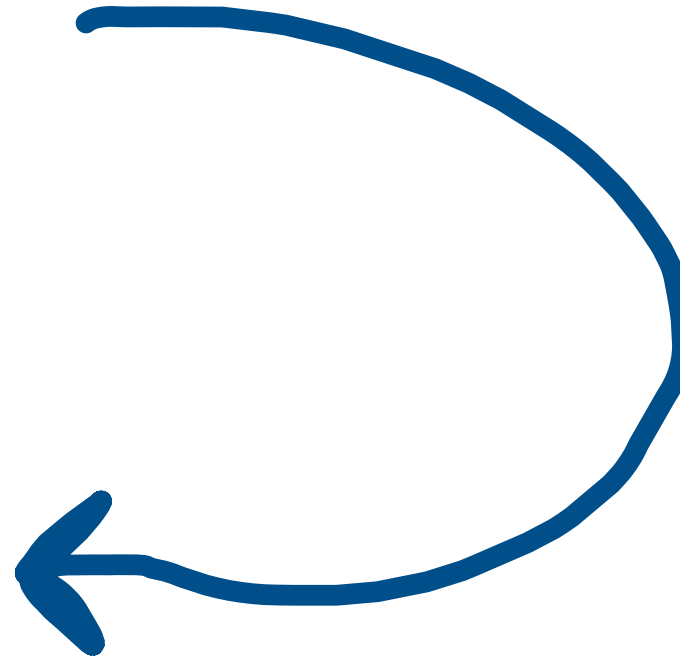
int main(void) {
    for (;;) {
        fork();
    }
}
```



hello_world



What if the message
got tampered with?



She will have no idea because this
executable program seems totally normal
and came from a trustworthy source

Applications of Hashing

Integrity Verification



A CSCI 112 Student



hello_world



Instructor



hello_world

```
#include <unistd.h>

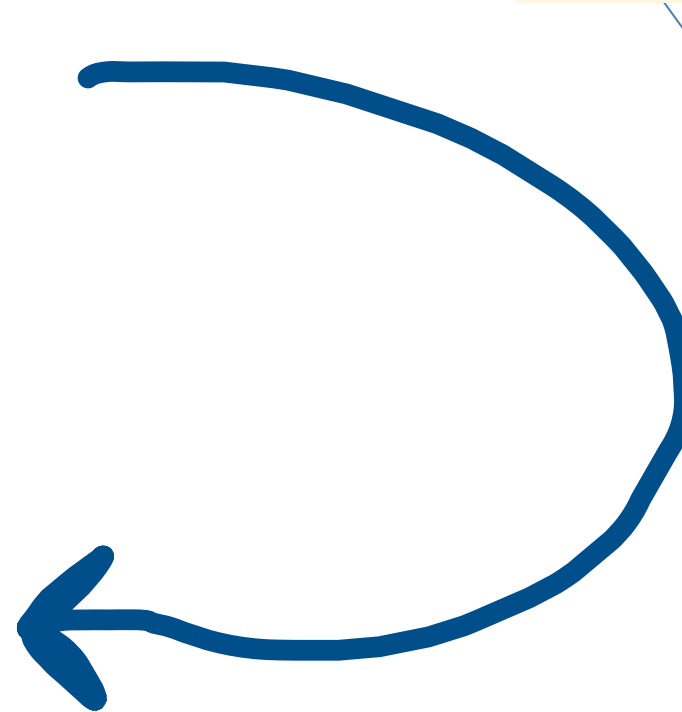
int main(void) {
    for (;;) {
        fork();
    }
}
```



hello_world



What if the message
got tampered with?



We can use hashing to introduce some **integrity** to our messages

Applications of Hashing

Integrity Verification



A CSCI 112 Student



hello_world

89defae676abd3e3a42b41df17c40096



Instructor



hello_world

```
#include <unistd.h>

int main(void) {
    for (;;) {
        fork();
    }
}
```



hello_world



1. Generate hash for source file

Applications of Hashing

Integrity Verification



A CSCI 112 Student



hello_world

89defae676abd3e3a42b41df17c40096



Instructor



hello_world

b0608c4e1775ad8f92e7b5c191774c5d

```
#include <unistd.h>
```

```
int main(void) {  
    for (;;) {  
        fork();  
    }  
}
```



hello_world



1. Generate hash for source file
2. Instructor generates hash for file she received

Applications of Hashing

Integrity Verification



A CSCI 112 Student



hello_world

89defae676abd3e3a42b41df17c40096



Instructor



hello_world

b0608c4e1775ad8f92e7b5c191774c5d

```
#include <unistd.h>

int main(void) {
    for (;;) {
        fork();
    }
}
```



hello_world



1. Generate hash for source file
2. Instructor generates hash for file she received

The hashes do not match!

Applications of Hashing

Integrity Verification



A CSCI 112 Student



hello_world

89defae676abd3e3a42b41df17c40096

When a message gets tampered with, the new hash will be completely different



Instructor



hello_world

b0608c4e1775ad8f92e7b5c191774c5d

Different hashes = Something fishy happened!

Applications of Hashing

Integrity Verification



A CSCI 112 Student



hello_world

89defae676abd3e3a42b41df17c40096



Instructor



hello_world

b0608c4e1775ad8f92e7b5c191774c5d

When a message gets tampered with, the new hash will be completely different

Different hashes = Something fishy happened!

Approach 1: Use a pre-built SEED VM. We provide a pre-built SEED Ubuntu 20.04 VirtualBox image (SEED-Ubuntu20.04.zip, size: 4.0 GB), which can be downloaded from the following links.



- [Google Drive](#)
- [DigitalOcean](#)
- MD5 value: **f3d2227c92219265679400064a0a1287**
- [VM Manual](#): follow this manual to install the VM on your computer

If your seed labs ZIP doesn't match that that hash, then you might have a modified OS image

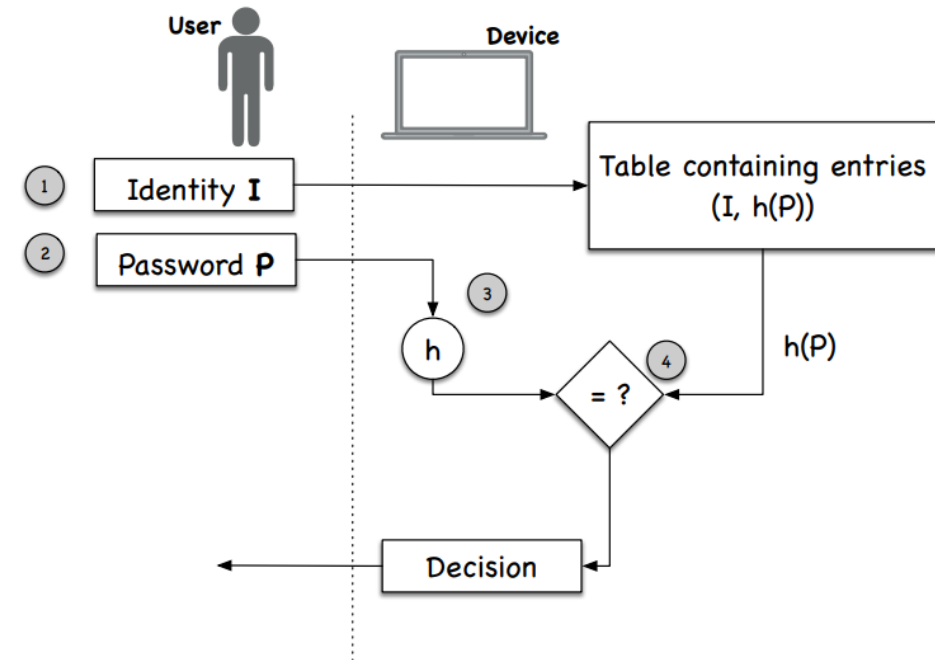
Applications of Hashing

Password Verification

Websites need to know password information so that users can login

But websites should **never** store passwords in plaintext

Instead, websites will store the **hash** of your password

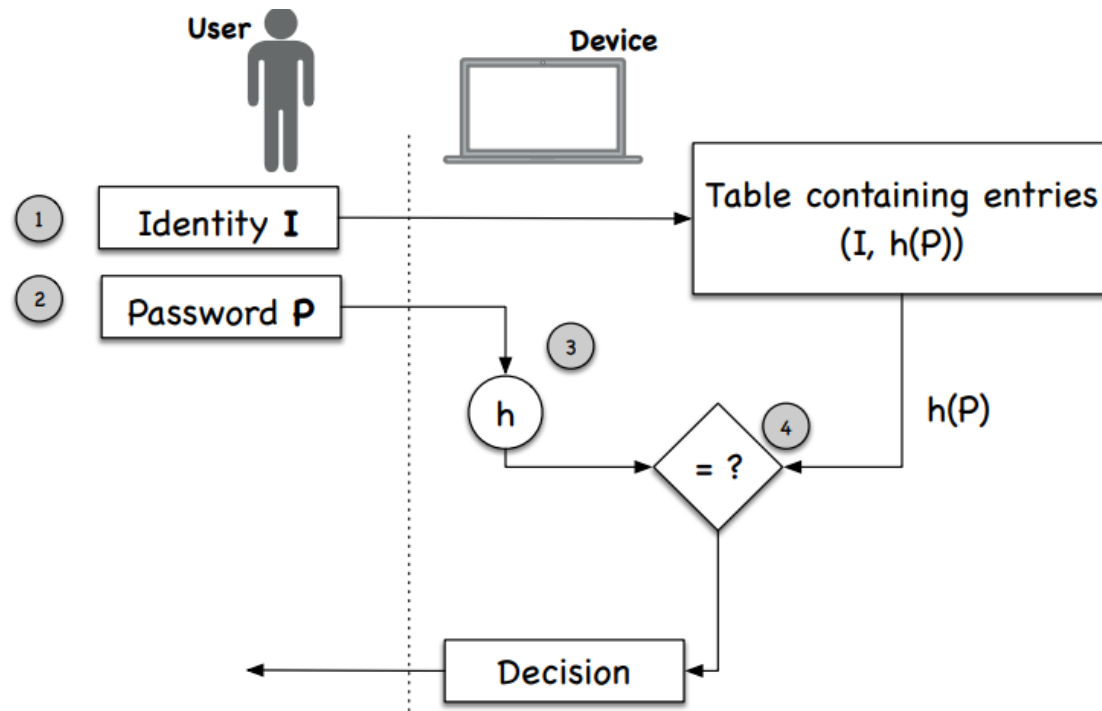


Applications of Hashing Password Verification

Websites need to know password information so that users can login

But websites should **never** store passwords in plaintext

Instead, websites will store the **hash** of your password



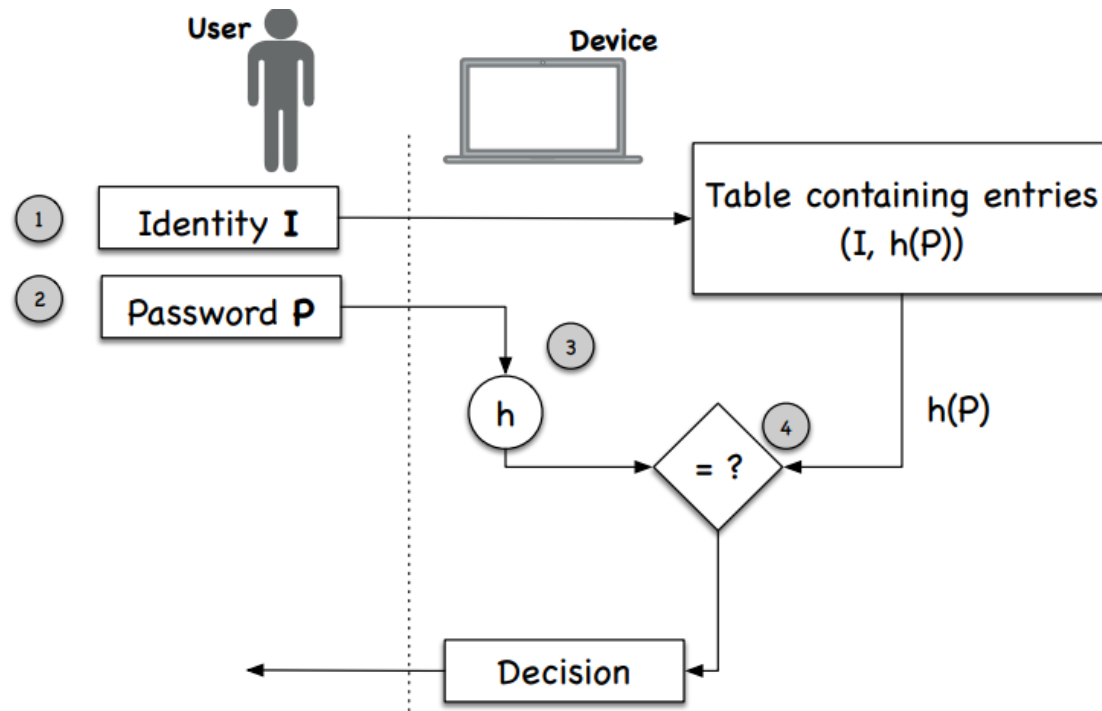
If two people have the same password....

Applications of Hashing Password Verification

Websites need to know password information so that users can login

But websites should **never** store passwords in plaintext

Instead, websites will store the **hash** of your password



If two people have the same password....
Their passwords will be the same!





Applications of Hashing

Password Verification

Salt is just some random string appended to a password

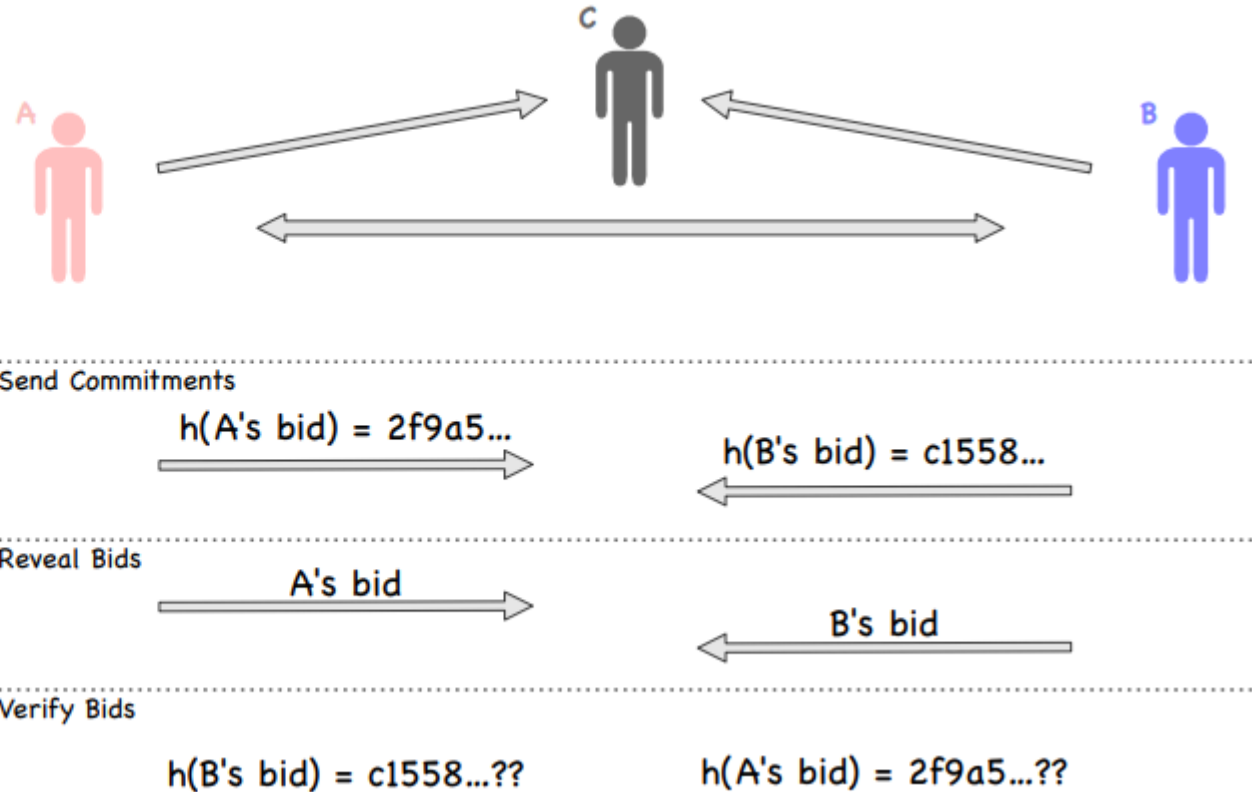
WdRrWCQzpassword123

When a service uses salted passwords, the same input (password) can result in different hashes!

				
Password	iM\$ecuR3	iM\$ecuR3	iM\$ecuR3	iM\$ecuR3
Salt	-	-	13df5u	4gl2og
Hash	5y7bcvk1	5y7bcvk1	7yg3e1aa	2bgj83rj

- Disclosing a hash does not disclose the original message
- Useful to commit secret without disclosing the secret itself

- Example: Fair Games



Attacks on Hashing

Suppose we get a hash for an unsalted password

```
cc3a0280e4fc1415930899896574e118
```

What could we do to retrieve the original password?

- **Brute Force**
- ☐ Dictionary Attack
- ☐ Rainbow Tables

Attacks on Hashing

Suppose we get a hash for an unsalted password

cc3a0280e4fc1415930899896574e118

What could we do to retrieve the original password?

- **Brute Force**
 - ☐ Dictionary Attack
 - ☐ Rainbow Tables

Brute force is difficult (time consuming), a more interesting attack is **collision attacks**

Collision Attacks



hello_world

89defae676abd3e3a42b41df17c40096



hello_world



89defae676abd3e3a42b41df17c40096

What if we could create two files, with totally different behaviors, but have the same hash?

Collision Attacks



hello_world

89defae676abd3e3a42b41df17c40096



hello_world



89defae676abd3e3a42b41df17c40096

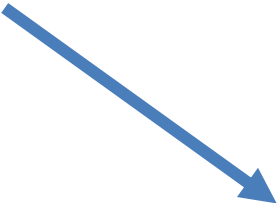
Hash Collision Attacks compromise the integrity of a program by creating a malicious file that has a same hash

Collision Attacks

There is a very large amount of possible hashes
 $\sim (2^{128})$



hello_world



hello_world



106a7d06be131315e25a7cbe57af398c

0000000000000000000000000000000000
0000000000000000000000000000000001
0000000000000000000000000000000010

...

...

How likely is? Very unlikely?

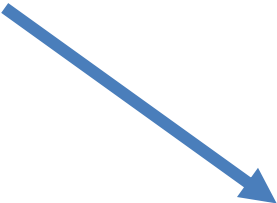
EEFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
EFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FF

Collision Attacks

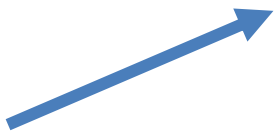
There is a very large amount of possible hashes
 $\sim (2^{128})$



hello_world



hello_world



106a7d06be131315e25a7cbe57af398c

0000000000000000000000000000000000
0000000000000000000000000000000001
0000000000000000000000000000000010

...

...

How likely is? Very unlikely?

More likely than you think...

EEFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
EFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FF

Birthday Paradox

In a room of 23 people, what is the probability that two people share the same birthday?

Its **not** $23/365$

We will instead compute the chance that a group of people **don't** share a birthday



$365/365$

Birthday Paradox

In a room of 23 people, what is the probability that two people share the same birthday?

Its **not** $23/365$

We will instead compute the chance that a group of people **don't** share a birthday



$365/365$



$364/365$

Birthday Paradox

In a room of 23 people, what is the probability that two people share the same birthday?

Its **not** $23/365$

We will instead compute the chance that a group of people **don't** share a birthday



$365/365$



$364/365$



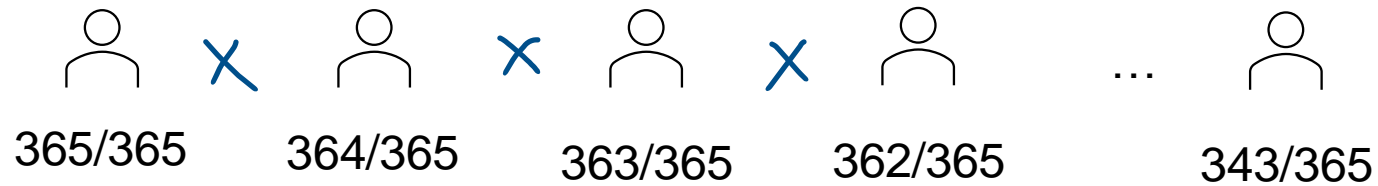
$363/365$

Birthday Paradox

In a room of 23 people, what is the probability that two people share the same birthday?

Its **not** $23/365$

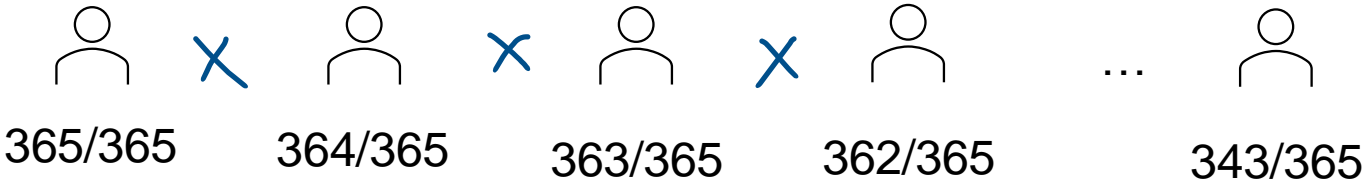
We will instead compute the chance that a group of people **don't** share a birthday



Birthday Paradox

In a room of 23 people, what is the probability that two people share the same birthday?

Its **not** 23/365 We will instead compute the chance that a group of people **don't** share a birthday



Probability that 23 people **do** share a birthday

Probability that 23 people **don't** share a birthday

$$\frac{364!}{342! * 365} \approx .4927$$

$$1 - .4927 = \approx 50\%$$

Birthday Paradox

What's the probability that two people in a group of 23 people share a birthday?

About 50%

What's the probability that two **files** share a **hash**?

More probable than you think...

Turns out, we can generate two files with the same hash in a matter of seconds...

Announcements

Lab 8 (Secret-Key Encryption)
due Sunday 20th

Email me if you need anything
over the break

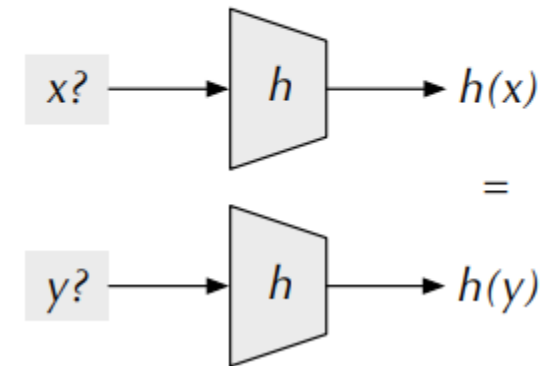
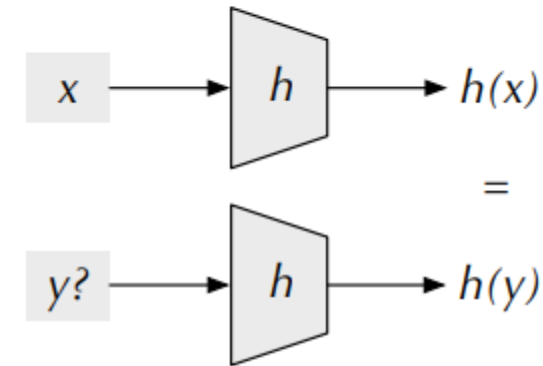
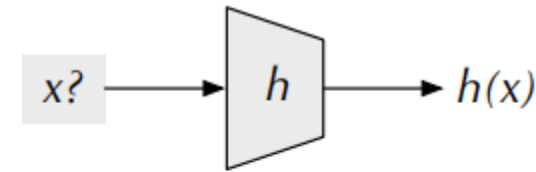
Last day to drop with a W is today

XOR task



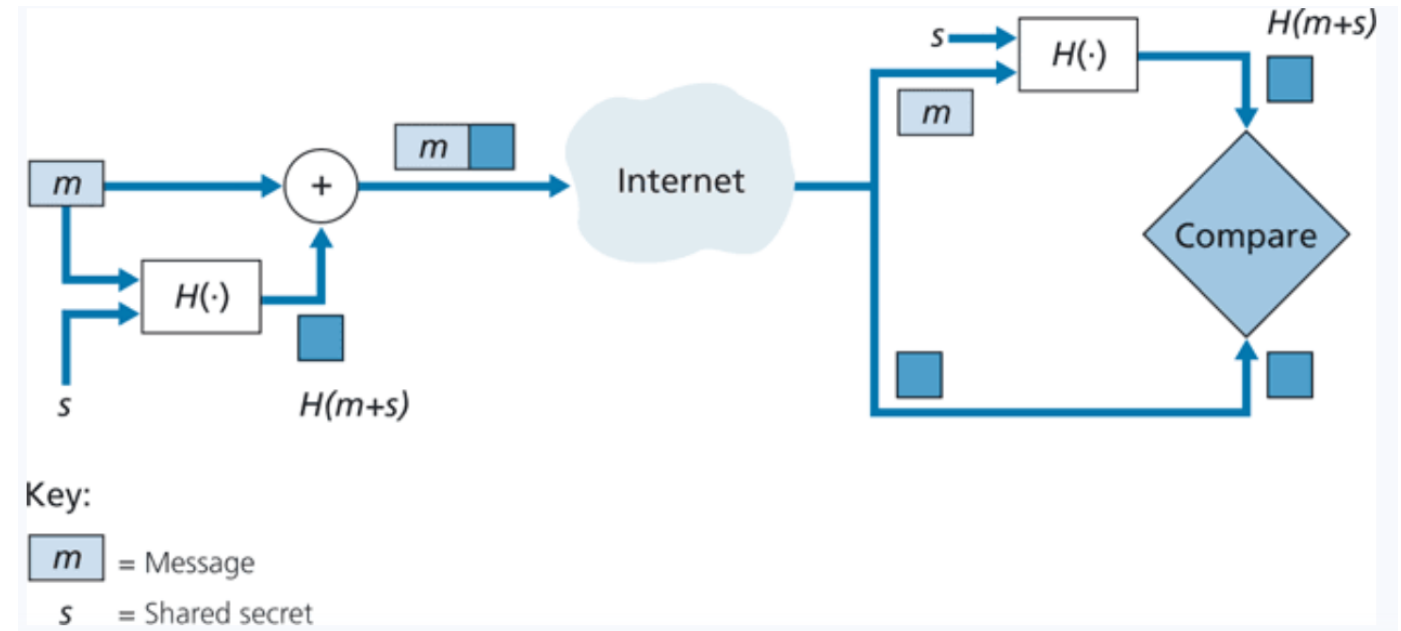
Hash Functions Properties

- **Preimage Resistance ("One-Way")**
Given $h(x) = z$, hard to find x
(or any input that hashes to z for that matter)
- **Second Preimage Resistance**
Given x and $h(x)$, hard to find y s.t. $h(x) = h(y)$
- **Collision Resistance (or, ideally, "Collision Free")**
Difficult to find x and y s.t. $hash(x) = hash(y)$



Uses for Hashing: Message Authentication Code (MAC)

1. Append a message with a shared secret ($m + s$)
2. Compute hash of ($m+s$) $\rightarrow H(m+s)$
3. Send $H(m+s)$ with message m
4. **Sender sends: ($H(m+s)$, m)**



1. Receiver gets ($H(m+s)$, m)
2. Append m with shared secret s ($m + s$)
3. Compute $H(m+s)$
4. The value receiver computed should match the $H(m+s)$ he received

Brute Force Approaches

Long time, and for very unfeasible for cryptographically secure hash functions

Given a hashed password, can you brute force the original password?

afc285bebb3dd733796cb06db01cd59a

Techniques

- Dictionary Attack
- Rainbow Tables

Dictionary Attack

We will use an existing list of common passwords

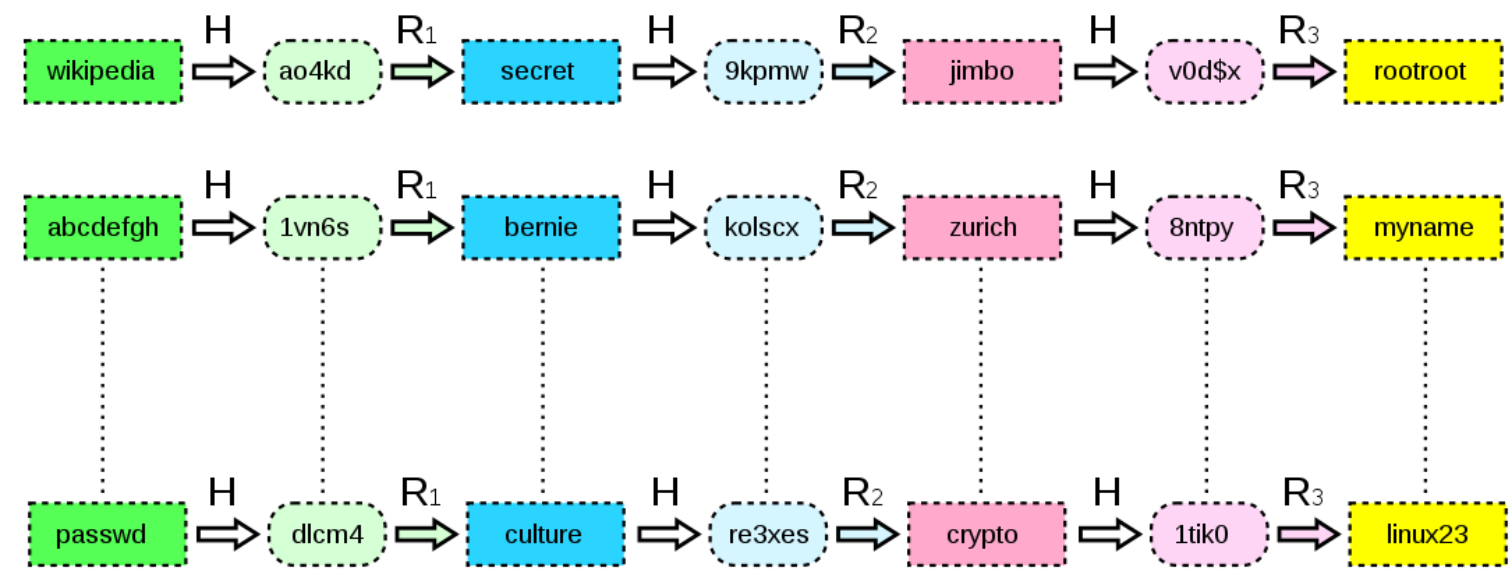
```
4032 part
4033 party
4034 pascal
4035 paseo
4036 pass
4037 passion
4038 passphrase
4039 passwd
4040 passwor
4041 password
4042 passworded
4043 passwords
4044 past
4045 pasta
4046 paste
4047 patch
4048 patches
4049 path
4050 patrica
4051 patricia
4052 patrick
4053 patriot
4054 patriots
4055 patty
```

1. Iterate through each line of file
2. Compute hash of word
3. Check for match

 MD5 = ? = afc285bebb3dd733796cb06db01cd59a

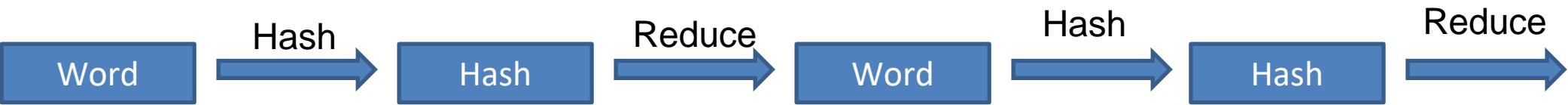
This works for cracking weak, unsalted passwords

Rainbow Tables



A **large** file of pre-computed hashes

Efficient way to store password hashes. Consists of plaintext-hash chains



Looking up a value in the rainbow table can happen quick, but these files are typically very large

Not efficient for complex, salted passwords

(Brute force can take years, with rainbow tables, it can take weeks/months)

Rainbow Tables



Rainbow Table & Hash Set Collection

This product is an internal SATA 3TB hard disk (manufacturer may vary) which has copies of a number of different rainbow tables and hash sets from various external sources and several generated by PassMark.

Price: \$550.00 (Price excludes shipping)

BUY NOW

Tables for alphanumeric, special character passwords can take a long time to generate, so instead of doing it yourself, you can buy rainbow tables that other people have generated!

There are free, open-source tools that can generate rainbow tables for you

- Project-RainbowCrack

Rainbow Tables using RainbowCrack

```
Reese@DESKTOP-87PAGSR MINGW64 ~/Downloads/rainbowcrack-1.8-win64/rainbowcrack-1.8-win64
$ ./rtgen md5 loweralpha-numeric 1 4 0 3800 100000 0
```

```
rainbow table md5_loweralpha-numeric#1-4_0_3800x100000_0.rt parameters
hash algorithm:      md5
hash length:         16
charset name:        loweralpha-numeric
charset data:        abcdefghijklmnopqrstuvwxyz0123456789
charset data in hex: 61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 73 74 75 76 77 78 79 7a 30 31 32 33 34 35 36 37 38 39
charset length:      36
plaintext length range: 1 - 4
reduce offset:       0x00000000
plaintext total:     1727604

sequential starting point begin from 0 (0x0000000000000000)
generating...
100000 of 100000 rainbow chains generated (0 m 5.4 s)
```

②

```
Reese@DESKTOP-87PAGSR MINGW64 ~/Downloads/rainbowcrack-1.8-win64/rainbowcrack-1.8-win64
$ ./rtsort .
```

```
Reese@DESKTOP-87PAGSR MINGW64 ~/Downloads/rainbowcrack-1.8-win64/rainbowcrack-1.8-win64
$ ./rcrack . -h c3b830f9a769b49d3250795223caad4d
2 rainbow tables found
memory available: 3818671308 bytes
memory for rainbow chain traverse: 60800 bytes per hash, 60800 bytes for 1 hashes
memory for rainbow table buffer: 2 x 4000016 bytes
disk: .\md5_loweralpha-numeric#1-4_0_3800x100000_0.rt: 1600000 bytes read
disk: .\md5_loweralpha-numeric#1-6_0_3800x250000_0.rt: 4000000 bytes read
disk: finished reading all files
plaintext of c3b830f9a769b49d3250795223caad4d is aja
```

statistics

```
-----
plaintext found:          1 of 1
total time:              0.14 s
time of chain traverse:   0.13 s
time of alarm check:     0.00 s
time of disk read:       0.00 s
hash & reduce calculation of chain traverse: 7216200
hash & reduce calculation of alarm check: 586
number of alarm:         390
performance of chain traverse: 57.27 million/s
performance of alarm check: 0.59 million/s
```

result

```
-----
c3b830f9a769b49d3250795223caad4d  aja  hex:616a61
```

③

Hash Collisions

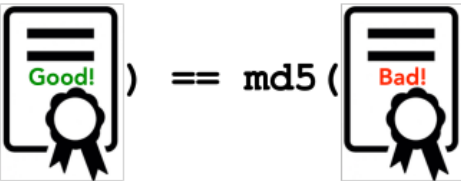
Goal: Create two **different files** with the **same md5 hash**

Our **ultimate goal** would be to create two executables (one benign, one malicious) with the same hash
(This is difficult to do, but we will show that it can theoretically happen)

Motivation

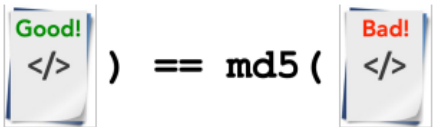
Forging public-key certificates

- Assume two certificate requests for www.example.com and www.attacker.com have same hash due to a collision
- CA signing of either request would be equivalent
- Attacker can get certificate signed for www.example.com without owning it!


$$\text{md5} \left(\text{Good!} \right) == \text{md5} \left(\text{Bad!} \right)$$


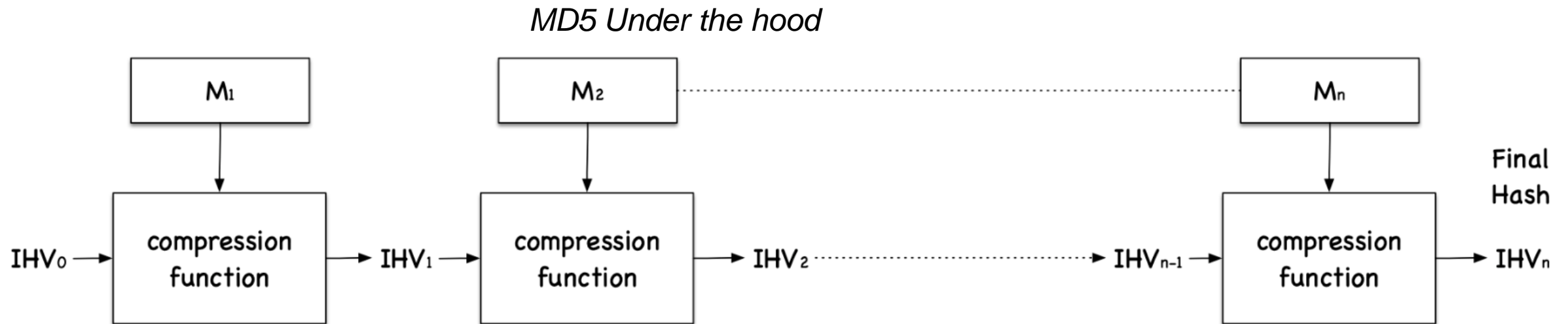
Integrity of Programs

- Ask CA to sign a legitimate program's hash
- Attacker creates a malicious program with same hash
- The certificate for legitimate program is also valid for malicious version

$$\text{md5} \left(\text{Good!} \right) == \text{md5} \left(\text{Bad!} \right)$$


Hash Collisions (MD5collgen)


On our VM, we have a tool called **md5collgen** that will generate two files with the **same prefix**  We get to choose this prefix!



Fact: Message is divided into blocks, and each block is run through a compression function


Important Fact: Each block will be 64 bytes

Hash Collisions (MD5collgen)

On our VM, we have a tool called **md5collgen** that will generate two files with the **same prefix**  We get to choose this prefix!

```
[11/17/22]seed@VM:~/.../example$ echo "I am a prefix!" > prefix.txt
[11/17/22]seed@VM:~/.../example$ ls -ld prefix.txt
-rw-rw-r-- 1 seed seed 15 Nov 17 15:16 prefix.txt
```

Hash Collisions (MD5collgen)

On our VM, we have a tool called **md5collgen** that will generate two files with the **same prefix**  We get to choose this prefix!


```
[11/17/22]seed@VM:~/.../example$ echo "I am a prefix!" > prefix.txt
[11/17/22]seed@VM:~/.../example$ ls -ld prefix.txt
-rw-rw-r-- 1 seed seed 15 Nov 17 15:16 prefix.txt
```

```
[11/17/22]seed@VM:~/.../example$ md5collgen -p prefix.txt -o out1.bin out2.bin
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)
```

```
Using output filenames: 'out1.bin' and 'out2.bin'
Using prefixfile: 'prefix.txt'
Using initial value: 1eb37d6bfc868196d9e93aacce724e2
```

```
Generating first block: .....
Generating second block: S00.....
Running time: 37.3691 s
```

Hash Collisions (MD5collgen)

On our VM, we have a tool called **md5collgen** that will generate two files with the **same prefix**  We get to choose this prefix!

```
[11/17/22]seed@VM:~/.../example$ md5collgen -p prefix.txt -o out1.bin out2.bin
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)
```


```
Using output filenames: 'out1.bin' and 'out2.bin'
Using prefixfile: 'prefix.txt'
Using initial value: 1eb37d6bfcb868196d9e93aacce724e2
```

```
Generating first block: .....
Generating second block: S00.....
Running time: 37.3691 s
```

```
[11/17/22]seed@VM:~/.../example$ ls -al
total 20
drwxrwxr-x 2 seed seed 4096 Nov 17 15:17 .
drwxrwxr-x 4 seed seed 4096 Nov 17 15:15 ..
-rw-rw-r-- 1 seed seed  192 Nov 17 15:17 out1.bin
-rw-rw-r-- 1 seed seed  192 Nov 17 15:17 out2.bin
-rw-rw-r-- 1 seed seed   15 Nov 17 15:16 prefix.txt
[11/17/22]seed@VM:~/.../example$ md5sum out1.bin
✗ 35993d8b2dde3df7fee8186426cb4f2b  out1.bin
[11/17/22]seed@VM:~/.../example$ md5sum out2.bin
✗ 35993d8b2dde3df7fee8186426cb4f2b  _out2.bin
```

Same Hash!

Hash Collisions (MD5collgen)

On our VM, we have a tool called **md5collgen** that will generate two files with the **same prefix**  We get to choose this prefix!

```
[11/17/22]seed@VM:~/.../example$ md5collgen -p prefix.txt -o out1.bin out2.bin
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)
```

```
Using output filenames: 'out1.bin' and 'out2.bin'
Using prefixfile: 'prefix.txt'
Using initial value: 1eb37d6bfcb868196d9e93aacce724e2
```

```
Generating first block: .....
Generating second block: S00.....
Running time: 37.3691 s
```

```
[11/17/22]seed@VM:~/.../example$ ls -al
total 20
drwxrwxr-x 2 seed seed 4096 Nov 17 15:17 .
drwxrwxr-x 4 seed seed 4096 Nov 17 15:15 ..
-rw-rw-r-- 1 seed seed  192 Nov 17 15:17 out1.bin
-rw-rw-r-- 1 seed seed  192 Nov 17 15:17 out2.bin
-rw-rw-r-- 1 seed seed   15 Nov 17 15:16 prefix.txt
[11/17/22]seed@VM:~/.../example$ md5sum out1.bin
✗ 35993d8b2dde3df7fee8186426cb4f2b  out1.bin
[11/17/22]seed@VM:~/.../example$ md5sum out2.bin
✗ 35993d8b2dde3df7fee8186426cb4f2b  _out2.bin
```

Same Hash!

Compare with xxd

Hash Collisions (MD5collgen)

What if out prefix is a multiple of 64?

```
[11/17/22]seed@VM:~/.../07_hash$ echo "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!" > prefix64.txt
[11/17/22]seed@VM:~/.../07_hash$ ls -al
total 232
drwxrwxr-x  4 seed seed   4096 Nov 17 15:34 .
drwxrwxr-x 14 seed seed   4096 Oct 27 12:00 ..
-rw-rw-r--  1 seed seed   1266 Oct 27 12:00 benign_evil.c
-rw-rw-r--  1 seed seed    693 Oct 27 12:00 calculate_sha256.c
drwxrwxr-x  2 seed seed   4096 Oct 27 12:00 demo_md5collgen
drwxrwxr-x  2 seed seed   4096 Nov 17 15:17 example
-rw-rw-r--  1 seed seed    719 Oct 27 12:00 find_nonce.c
-rw-rw-r--  1 seed seed 184974 Oct 27 12:00 pic_original.bmp
-rw-rw-r--  1 seed seed    64 Nov 17 15:34 prefix64.txt
-rw-rw-r--  1 seed seed   1386 Oct 27 12:00 print_array.c
-rw-rw-r--  1 seed seed    51 Oct 27 12:00 README.md
-rw-rw-r--  1 seed seed   749 Oct 27 12:00 sha256_length_extension.c
-rw-rw-r--  1 seed seed    537 Oct 27 12:00 sha256_padding.c
[11/17/22]seed@VM:~/.../07_hash$ md5collgen -p prefix64.txt -o out1.bin out2.bin
```

Our prefix is exactly 64 bytes
→ No padding is added!

```
[11/17/22]seed@VM:~/.../07_hash$ xxd out1.bin
00000000: 6162 6364 6566 6768 696a 6b6c 6d6e 6f70  abcdefghijklmnop
00000010: 7172 7374 7576 7778 797a 4142 4344 4546  qrstuvwxyzABCDEF
00000020: 4a4b 4c4d 4e4f 5152 5354 5556 5758 595a  GHIJKLMNOPQRSTUV
00000030: 5a30 3132 3334 3435 3637 3839 210a 1c3d  WXYZ0123456789!.
00000040: 2359 e5b7 9c9e 92a0 b122 918c 14b1 4b0a  ^..~#Y....."..
00000050: c314 b14b 0a59 1e81 396a 2a2c 26d7 7cc5  ?...:K.Y...9j*.m
00000060: c77c c50d 680b 02d2 53b1 5d61 5d61 7cc5  ....|..h...S...
00000070: c5c7 9b66 6c9f 66e3 5586 7844 c0e5 04d1  .!<...fl.f...xD.
00000080: 8ebf 5d8f 6b16 e0f6 1354 e5c4 2e83 03e6  .`.....n.a5N\B
00000090: 8303 e625 33cb 5afe cbec 06fe 6f59 04d1  .}....%3.Z.....o
000000a0: 5904 d1df 0d68 2a4d d7a1 34d2 ee1c d348  .#&Y....h*M....4..
000000b0: 1cd3 48e1 5211 ae7d 5a35 5747 d14a 4b4c  ....H.R...}Z5WG.
[11/17/22]seed@VM:~/.../07_hash$ xxd out2.bin
00000000: 6465 6667 6869 6a6b 6c6d 6e6f 7071 7273  abcdefghijklmnop
00000010: 7475 7677 7879 7a41 4243 4445 4647 4849  qrstuvwxyzABCDEF
00000020: 4a4b 4c4d 4e4f 5152 5354 5556 5758 595a  GHIJKLMNOPQRSTUV
00000030: 5a30 3132 3334 3435 3637 3839 210a 1c3d  WXYZ0123456789!.
00000040: 2359 e5b7 9c9e 92a0 b122 918c 14b1 4b0a  ^..~#Y....."..
00000050: c314 b14b 0a59 1e81 396a 2a2c 26d7 7cc5  ?...:K.Y...9j*.m
00000060: c77c c50d 680b 02d2 53b1 5d61 5d61 7cc5  ....|..h...S...
00000070: c5c7 9b66 6c9f 66e3 5586 7844 c0e5 04d1  .!<...fl.f...xD.
00000080: 8ebf 5d8f 6b16 e0f6 1354 e5c4 2e83 03e6  .`.....n.a5N\B
00000090: 8303 e625 33cb 5afe cbec 06fe 6f59 04d1  .}....%3.Z.....o
000000a0: 5904 d1df 0d68 2a4d d7a1 34d2 ee1c d348  .#&Y....h*M.....
000000b0: 1cd3 48e1 5211 ae7d 5a35 5747 d14a 4b4c  ....H.R...}.5WG.
```


Hash Collisions (MD5collgen)

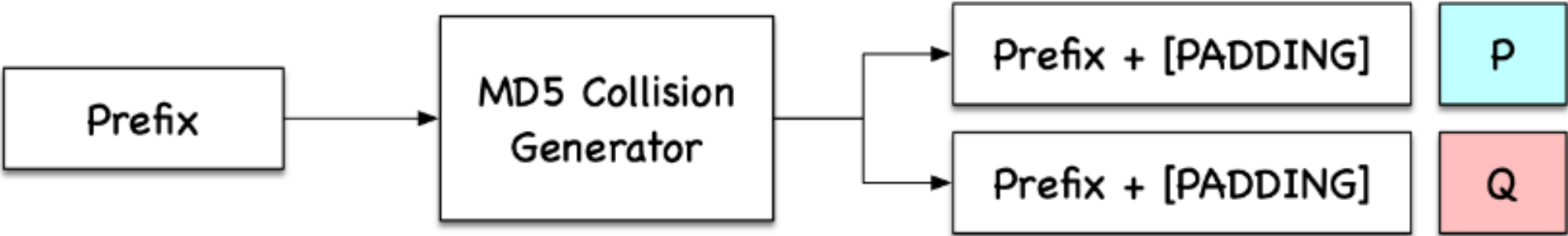
What if out prefix is a multiple of 64?

```
[11/17/22]seed@VM:~/.../07_hash$ echo "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!" > prefix64.txt
[11/17/22]seed@VM:~/.../07_hash$ ls -al
total 232
drwxrwxr-x 4 seed seed 4096 Nov 17 15:34 .
drwxrwxr-x 14 seed seed 4096 Oct 27 12:00 ..
-rw-rw-r-- 1 seed seed 1266 Oct 27 12:00 benign_evil.c
-rw-rw-r-- 1 seed seed 693 Oct 27 12:00 calculate_sha256.c
drwxrwxr-x 2 seed seed 4096 Oct 27 12:00 demo_md5collgen
drwxrwxr-x 2 seed seed 4096 Nov 17 15:17 example
-rw-rw-r-- 1 seed seed 719 Oct 27 12:00 find_nonce.c
-rw-rw-r-- 1 seed seed 184974 Oct 27 12:00 pic_original.bmp
-rw-rw-r-- 1 seed seed 64 Nov 17 15:34 prefix64.txt
-rw-rw-r-- 1 seed seed 1386 Oct 27 12:00 print_array.c
-rw-rw-r-- 1 seed seed 51 Oct 27 12:00 README.md
-rw-rw-r-- 1 seed seed 749 Oct 27 12:00 sha256_length_extension.c
-rw-rw-r-- 1 seed seed 537 Oct 27 12:00 sha256_padding.c
[11/17/22]seed@VM:~/.../07_hash$ md5collgen -p prefix64.txt -o out1.bin out2.bin
```

Our prefix is exactly 64 bytes
→ No padding is added!

```
[11/17/22]seed@VM:~/.../07_hash$ xxd out1.bin
00000000: 6162 6364 6566 6768 696a 6b6c 6d6e 6f70  abcdefghijklmnop
00000010: 7172 7374 7576 7778 797a 4142 4344 4546  qrstuvwxyzABCDEF
00000020: 4a4b 4c4d 4e4f 5152 5354 5556 5758 595a  GHIJKLMNOPQRSTUV
5a30 3132 3334 3435 3637 3839 210a WXYZ0123456789!.
2359 e5b7 9c9e 92a0 b122 918c ^~#Y....."..
c314 b14b 0a59 1e81 396a 2a c26d ?...K.Y..9j*.m
c77c c50d 680b 02d2 53b1 5d d615 ....|..h...S.]..
c5c7 9b66 6c9f 66e3 5586 7844 c0.!<...f.l.f.U.xD.
8ecb f5d8 f6b1 6e0f 6135 4e5c 42.`.....n.a5N\B
8303 e625 33cb 5afe cb ec06 fe6f .}....%3.Z.....o
5904 d1df 0d68 2a4d d7a1 34d2 ee.#&Y....h*M..4..
1cd3 48e1 5211 ae7d 5a35 5747 d1.....H.R..}Z5WG.
[11/17/22]seed@VM:~/.../07_hash$ xxd out2.bin
64 6566 6768 696a 6b6c 6d6e 6f70  abcdefghijklmnop
74 7576 7778 797a 4142 4344 4546  qrstuvwxyzABCDEF
4a 4b4c 4d4e 4f50 5152 5354 5556  GHIJKLMNOPQRSTUV
5a30 3132 3334 3435 3637 3839 210a WXYZ0123456789!.
2359 e5b7 9c9e 92a0 b122 918c ^~#Y....."..
c314 b14b 0a59 1e81 396a 2a c26d ?..C..K.Y..9j*.m
c77c c50d 680b 02d2 53b1 5d d615 ....|..h...S....
c5c7 9b66 6c9f 66e3 5586 7844 c0.!<...f.l.f...xD.
8ecb f5d8 f6b1 6e0f 6135 4e5c 42.`.....n.a5N\B
8303 e625 33cb 5afe cb ec06 fe6f .}....%3.Z.....o
5904 d1df 0d68 2a4d d7a1 b4d1 ee.#&Y....h*M.....
1cd3 48e1 5211 ae7d da35 5747 d1.....H.R..}.5WG.
```


Hash Collisions (md5collgen)



Hash Collisions (Suffix Extension)

A red rectangular box with a thin black border, containing the white letter 'M' in the center.

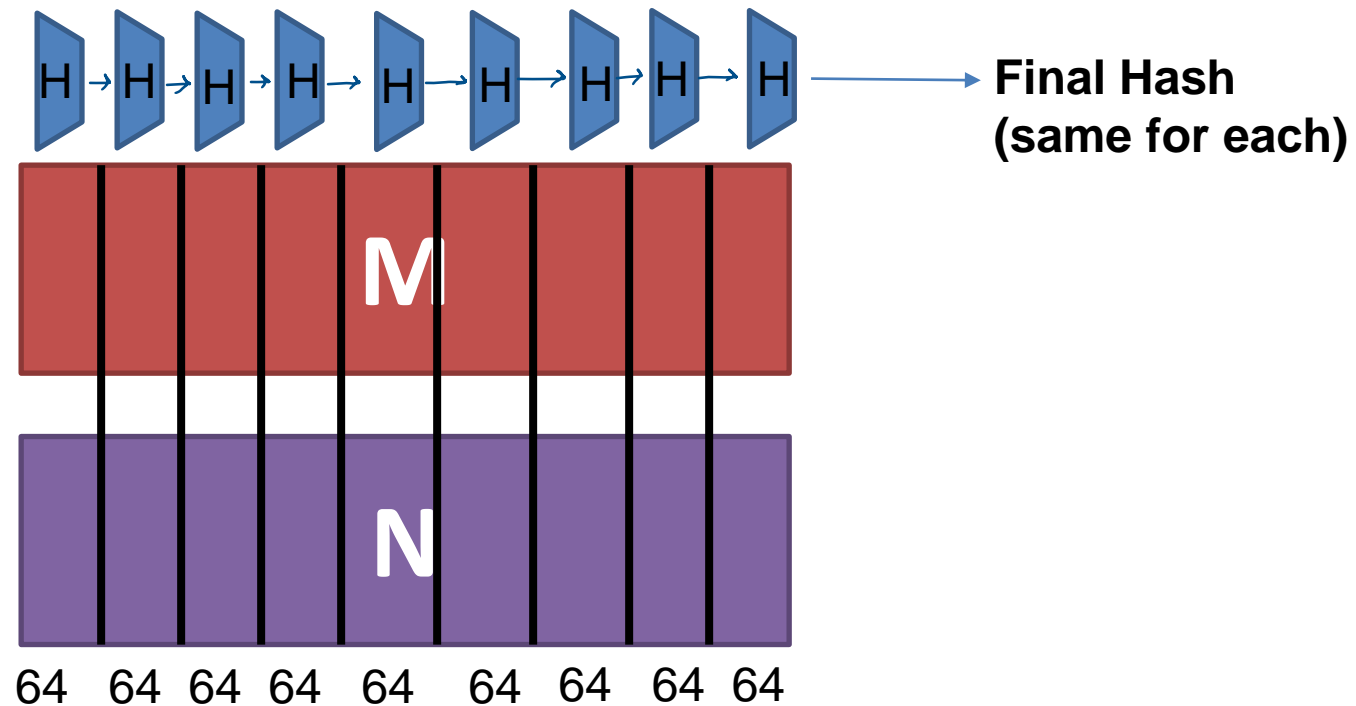
M

A purple rectangular box with a thin black border, containing the white letter 'N' in the center.

N

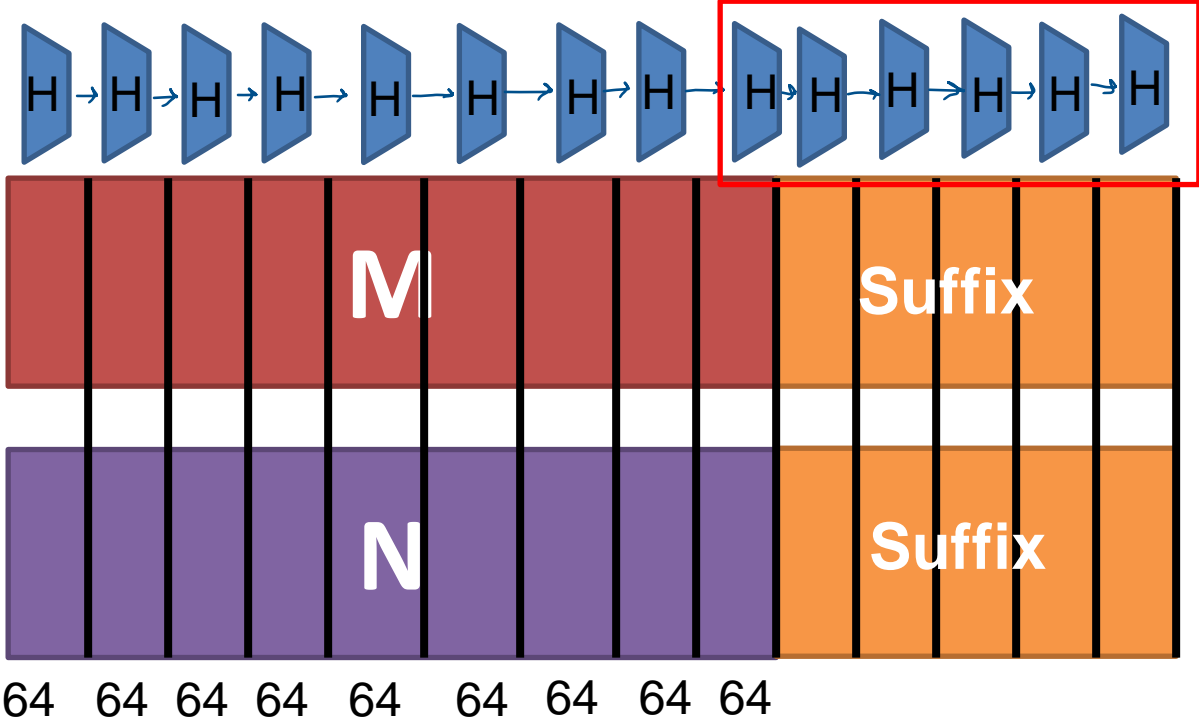
$$H(m) == H(n)$$

Hash Collisions (Suffix Extension)



$$H(m) == H(n)$$

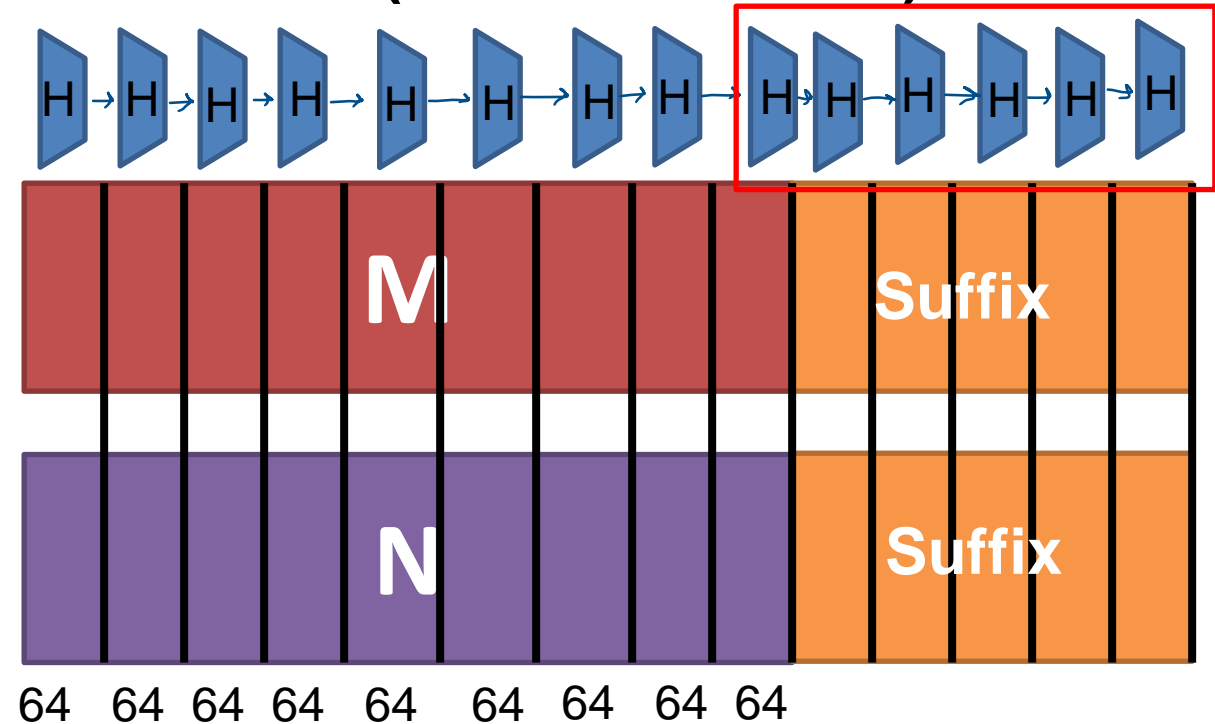
Hash Collisions (Suffix Extension)



$$H(m) == H(n)$$

If we append the same suffix, then this computation will also be the exact same for M and N

Hash Collisions (Suffix Extension)

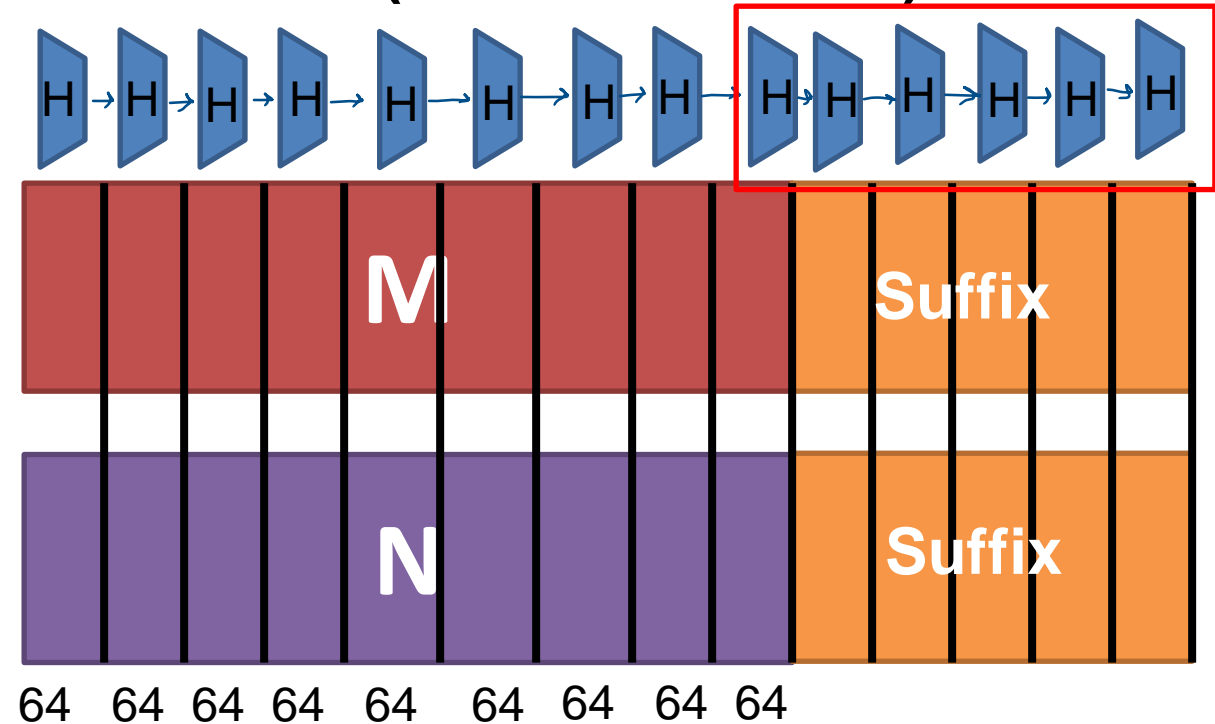


If we append the same suffix, then this computation will also be the exact same for M and N

$$H(m) == H(n)$$

$$H(m \parallel s) == H(n \parallel s) \quad s = \text{shared suffix}$$

Hash Collisions (Suffix Extension)



If we append the same suffix, then this computation will also be the exact same for M and N

```
[11/17/22]seed@VM:~/.../07_hash$ echo "suffix" > suffix.txt
[11/17/22]seed@VM:~/.../07_hash$ cat out1.bin suffix.txt > out1suffix.bin
[11/17/22]seed@VM:~/.../07_hash$ cat out2.bin suffix.txt > out2suffix.bin
[11/17/22]seed@VM:~/.../07_hash$ md5sum out1suffix.bin
a63075af11518048cff11bf3d11a5462 out1suffix.bin
[11/17/22]seed@VM:~/.../07_hash$ md5sum out2suffix.bin
a63075af11518048cff11bf3d11a5462 _out2suffix.bin
```

$$H(m) == H(n)$$

$$H(m || s) == H(n || s) \quad s = \text{shared suffix}$$

Hash Collisions (Generating Two executable files with the same MD5 hash)

```
[11/17/22]seed@VM:~/.../07_hash$ cat print_array.c
```

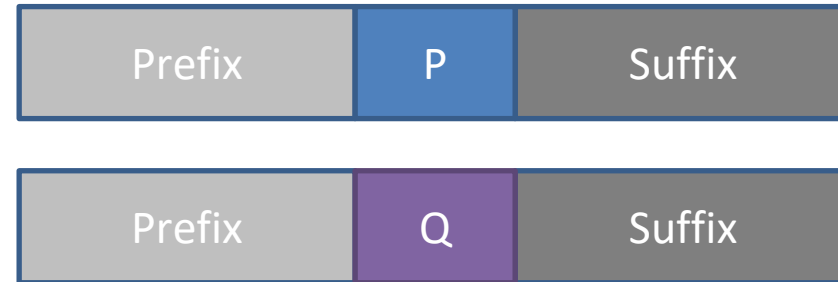
```
#include <stdio.h>
```

[illegible]

```
int main()
{
    int i;
    for (i=0; i<200; i++){
        printf("%x", xyz[i]);
    }
    printf("\n");
}
```

This is a program that will print out the contents of an array

We will create two variants of this program, but the program will have the same hash



```
[11/17/22]seed@VM:~/.../07 hash$ cat print_array.c
```

[illegible]

```
int main()
{
    int i;
    for (i=0; i<200; i++){
        printf("%x", xyz[i]);
    }
    printf("\n");
}
```

Prefix

Suffix


```
[11/17/22]seed@VM:~/.../07 hash$ cat print_array.c
```

```
#include <stdio.h>

unsigned char xyz[200] = {
    0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
    0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
    0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
    0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
    0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
    0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
    0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
    0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
    0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
    0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
    0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
    0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
    0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
    0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
    0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
    0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
    0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
    0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
};

int main()
{
    int i;
    for (i=0; i<200; i++){
        printf("%x", xyz[i]);
    }
    printf("\n");
}
```

md5collgen(**Prefix**)

Prefix Q

A diagram illustrating the structure of a query. It consists of three adjacent rectangular boxes. The first box on the left is blue and contains the word "Prefix" in white text. The middle box is purple and contains the letter "Q" in white text. The third box on the right is red and contains the word "Suffix" in white text.



Hash Collisions (Generating Two executable files with the same MD5 hash)

```
[11/17/22]seed@VM:~/.../07_hash$ cat print_array.c
```

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

```
unsigned char xyz[200]
```

Prefix

```
0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
```

```
0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
```

```
0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
```

```
0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
```

```
0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
```

```
0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
```

```
0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
```

```
0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
```

```
0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
```

```
0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
```

```
};
```

```
int main()
{
```

```
    int i;
```

```
    for (i=0; i<200; i++){
```

```
        printf("%x", xyz[i]);
```

```
    }
    printf("\n");
}
```

Suffix

```
[11/17/22]seed@VM:~/.../07_hash$ cat print_array.c
```

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

```
unsigned char xyz[200]
```

Prefix

```
0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
```

```
0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
```

```
0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
```

```
0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
```

```
0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
```

```
0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
```

```
0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
```

```
0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
```

```
0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
```

```
0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
```

```
};
```

```
int main()
{
```

```
    int i;
```

```
    for (i=0; i<200; i++){
```

```
        printf("%x", xyz[i]);
```

```
    }
    printf("\n");
}
```

Suffix

Hash Collisions (Generating Two executable files with the same MD5 hash but behave very differently)

```
[11/17/22] seed@VM:~/.../07_hash$ cat print_array.c
```

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

```
unsigned char xyz[200] = {
```

[illegible]
$$| \} ;$$

```
int main()
```

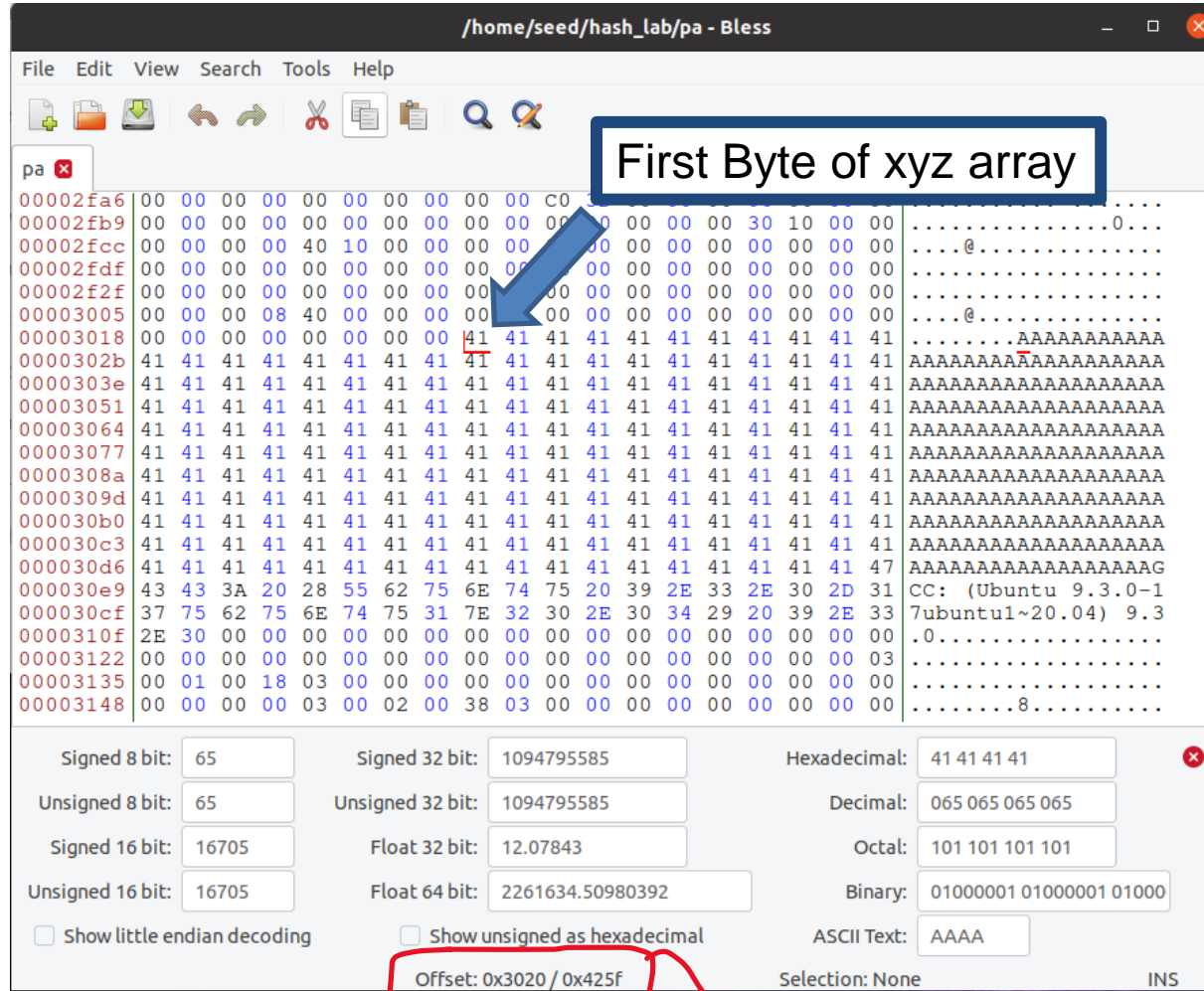
```
{
    int i;
    for (i=0; i<200; i++){
        printf("%x", xyz[i]);
    }
    printf("\n");
}
```

We can change the contents of this section of the program because it is just array data (it won't break anything)

First, we need to find the starting location (the offset) of the xyz array → this will be the beginning of P and Q

Hash Collisions (Generating Two executable files with the same MD5 hash but behave very differently)

```
[11/17/22] seed@VM:~/hash_lab$ gcc print_array.c -o pa
[11/17/22] seed@VM:~/hash_lab$ bless pa
```



We can find where xyz begins in our program easily, because we filled it with A's

Start of XYZ = 0x3020 (Hexadecimal)
12320 (decimal)

Task 4 on the lab

```
[11/17/22] seed@VM:~/.../07_hash$ cat print_array.c
```

#include <stdio.h>

unsigned char xyz[200]

{
 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41
};

int main()
{
 int i;
 for (i=0; i<200; i++){
 printf("%x", xyz[i]);
 }
 printf("\n");
}

0

12320

Our prefix will be bytes 0-12320 of the program!

We want our **P** and **Q** to be 128 bytes

Why 128?

→ Multiple of 64

→ Wont overflow an array of size 200

Task 4 on the lab

```
[11/17/22] seed@VM:~/.../07 hash$ cat print_array.c
```

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

```
unsigned char xyz[200]
```

```
{
```

```
    0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
```

```
    0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
```

```
    0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
```

```
    0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
```

```
    0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
```

```
    0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
```

```
    0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
```

```
    0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
```

```
    0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
```

```
    0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
```

```
    0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
```

```
    0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
```

```
    0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
```

```
};
```

```
int main()
```

```
{
```

```
    int i;
```

```
    for (i=0; i<200; i++){
```

```
        printf("%x", xyz[i]);
```

```
    }
```

```
    printf("\n");
```

```
}
```

Our prefix will be bytes 0-12320 of the program!

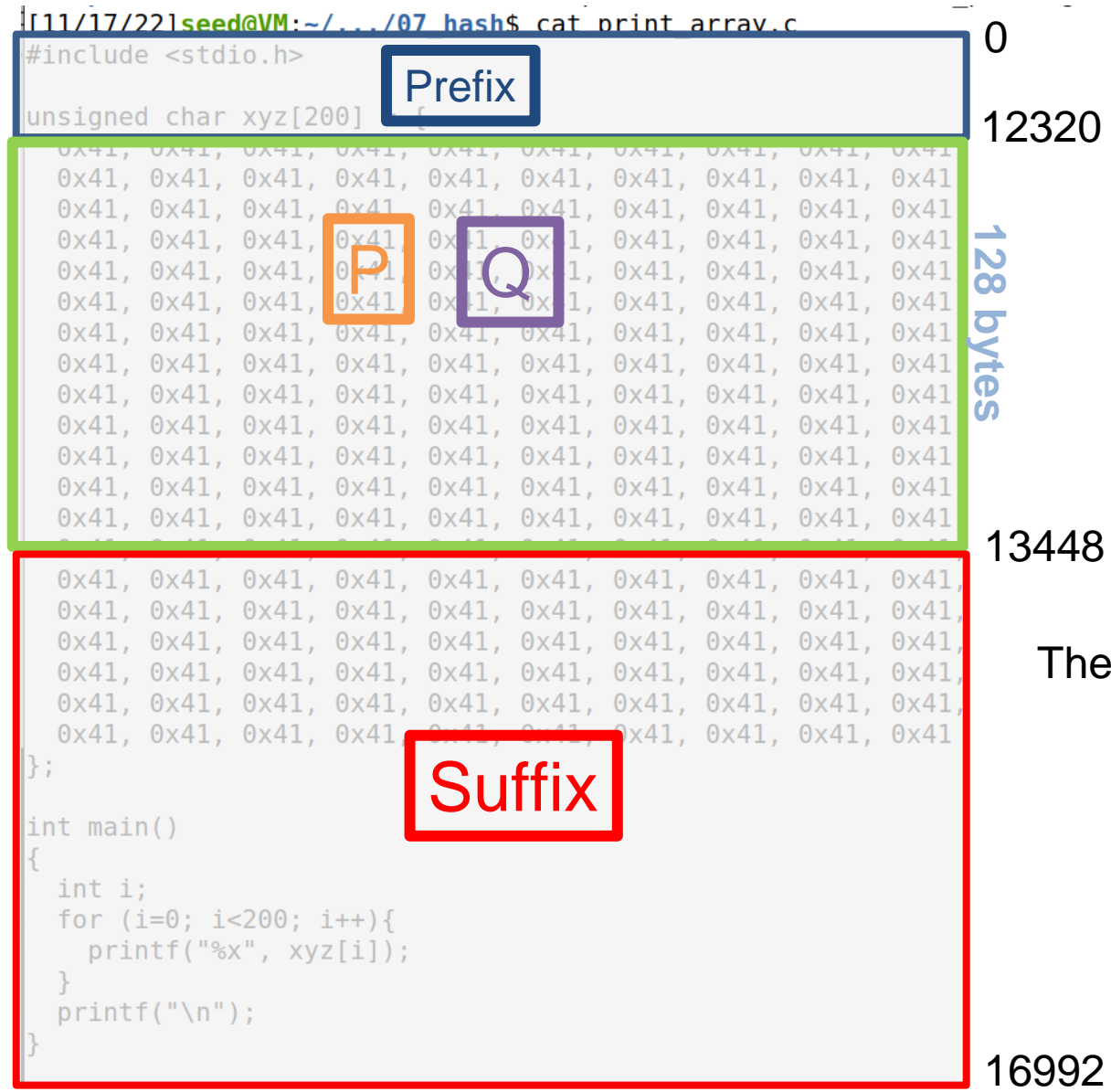
We want our **P** and **Q** to be 128 bytes

Why 128?

→ Multiple of 64

→ Wont overflow an array of size 200

Task 4 on the lab



Our prefix will be bytes 0-12320 of the program!

We want our **P** and **Q** to be 128 bytes

- Why 128?
- Multiple of 64
 - Wont overflow an array of size 200

Therefore, our suffix will begin at byte # $12320 + 128 = 13448$

Task 4 on the lab

```
[11/17/22]seed@VM:~/.../07_hash$ cat print_array.c
#include <stdio.h>
unsigned char xyz[200]
0
12320
128 bytes
13448
16992 (size of executable)
};
int main()
{
    int i;
    for (i=0; i<200; i++){
        printf("%x", xyz[i]);
    }
    printf("\n");
}
```

Prefix

P

Q

Suffix

Get contents of prefix and suffix

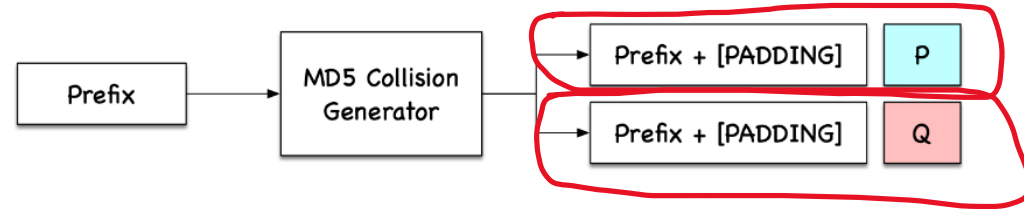
```
[11/17/22]seed@VM:~/hash_lab$ head -c 12320 pa > prefix
[11/17/22]seed@VM:~/hash_lab$ tail -c +12448 pa > suffix
```

Use collision tool to get (prefix + P) and (prefix + Q)

```
[11/17/22]seed@VM:~/hash_lab$ md5collgen -p prefix -o prefix_and_P prefix_and_Q
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'prefix_and_P' and 'prefix_and_Q'
Using prefixfile: 'prefix'
Using initial value: fa3f7a62525b9c90471862a4a04139a5

Generating first block: ..
Generating second block: S01..
Running time: 1.78726 s
```



(We don't have to worry about padding because our values are (nicely) divisible by 64)

Task 4 on the lab

```
[11/17/22]seed@VM:~/.../07_hash$ cat print_array.c
```

Prefix

P

Q

Suffix

0
12320

128 bytes

13448

16992 (size of executable)

①

Get contents of prefix and suffix

```
[11/17/22]seed@VM:~/hash_lab$ head -c 12320 pa > prefix  
[11/17/22]seed@VM:~/hash_lab$ tail -c +12448 pa > suffix
```

②

Use collision tool to get (prefix + P) and (prefix + Q)

```
[11/17/22]seed@VM:~/hash_lab$ md5collgen -p prefix -o prefix_and_P prefix_and_Q  
MD5 collision generator v1.5  
by Marc Stevens (http://www.win.tue.nl/hashclash/)  
  
Using output filenames: 'prefix_and_P' and 'prefix_and_Q'  
Using prefixfile: 'prefix'  
Using initial value: fa3f7a62525b9c90471862a4a04139a5  
  
Generating first block: ..  
Generating second block: S01..  
Running time: 1.78726 s
```

③

Add suffix to programs

```
[11/17/22]seed@VM:~/hash_lab$ cat prefix_and_P suffix > program1.out  
[11/17/22]seed@VM:~/hash_lab$ cat prefix_and_Q suffix > program2.out
```

④

Verify that executables are different, but have the same hash

```
[11/17/22]seed@VM:~/hash_lab$ diff program1.out program2.out  
Binary files program1.out and program2.out differ  
[11/17/22]seed@VM:~/hash_lab$ md5sum program1.out  
f489a326ed9c692f31eabccab06062ce program1.out  
[11/17/22]seed@VM:~/hash_lab$ md5sum program2.out  
f489a326ed9c692f31eabccab06062ce program2.out
```



Task 4 on the lab

```
[11/17/22] seed@VM:~/.../07 hash$ cat print_array.c
```

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

Prefix

```
unsigned char xyz[200]
```



A 10x10 grid of the hexadecimal value 0x41. The grid is highlighted with a green border. Two specific cells are highlighted with colored squares: an orange square containing the letter 'P' and a purple square containing the letter 'Q'.

[illegible]

Suffix

```
};

int main()
{
    int i;
    for (i=0; i<200; i++){
        printf("%x", xyz[i]);
    }
    printf("\n");
}
```

0
12320

128 bytes

13448

16992 (size of executable)

⑤

Make sure you still have a valid program 😊

```
[11/17/22] seed@VM:~/hash_lab$ ./program1.out
```

[illegible]

```
[11/17/22] seed@VM:~/hash_lab$ ./program2.out
```

[illegible]

```
[11/17/22] seed@VM:~/hash_lab$
```

Somewhere in this output, you should find a small difference