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

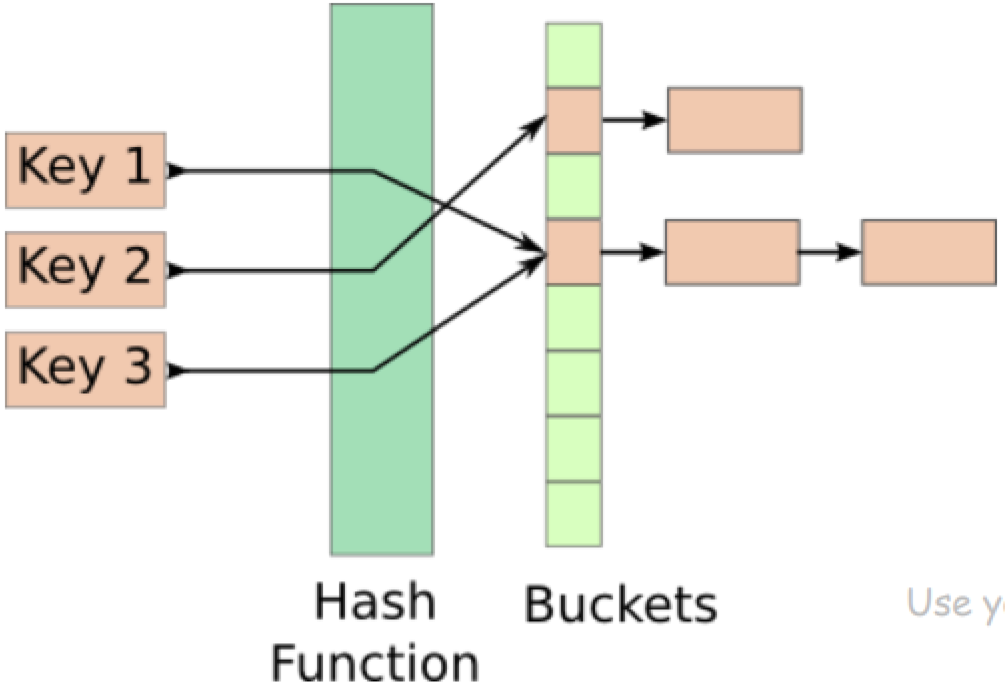
ENCRYPTION TYPES
NONE
ECB
Corporate needs you to find the differences between this picture and this picture.
They're the same picture.
imgflip.com

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 mod 100**



How many buckets?
What to do if two keys map to the same bucket?

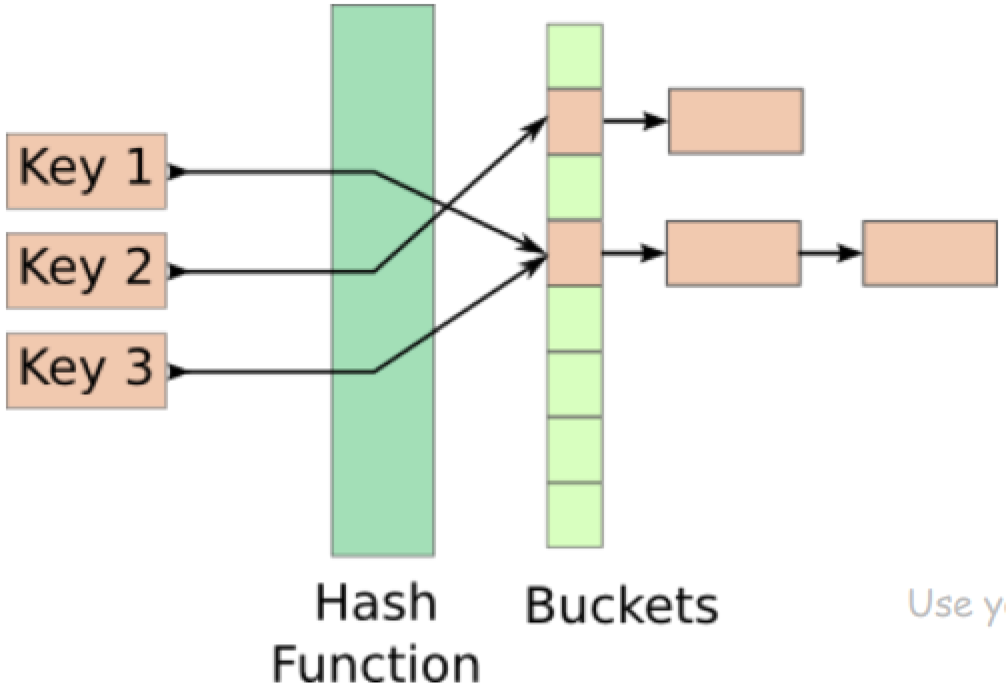
Collisions happen...
Use your favorite collision resolution technique
(open addressing, chaining, etc.)

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 mod 100**



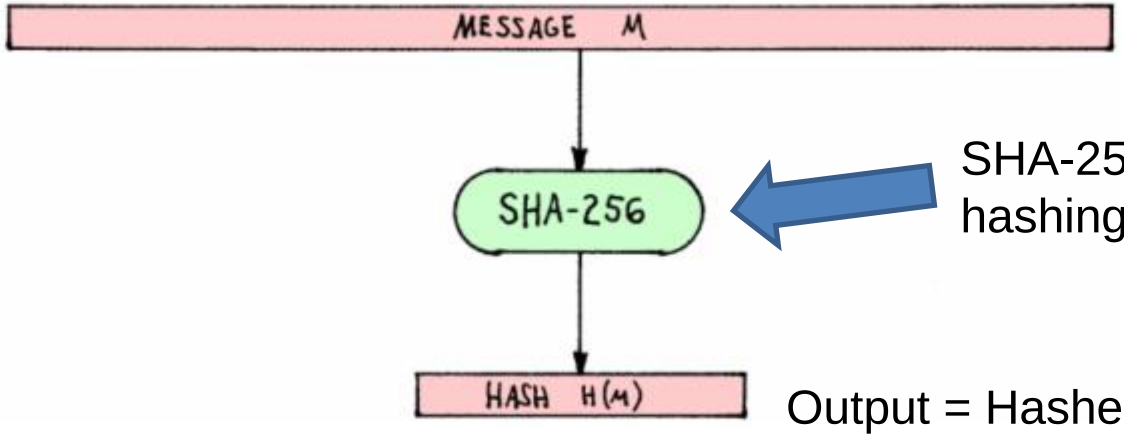
How many buckets?
What to do if two keys map to the same bucket?

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



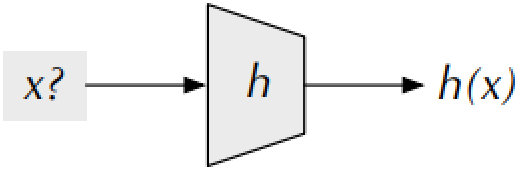
Input = Message M

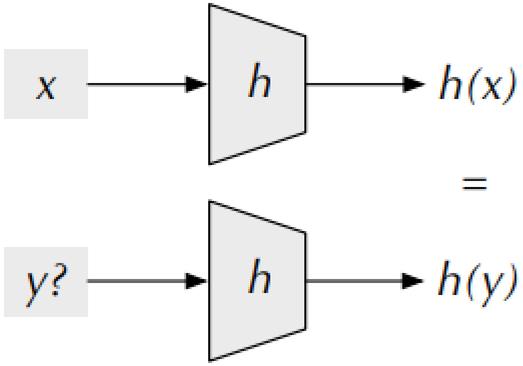
SHA-256 = A cryptographically secure hashing function

Output = Hashed Message M

Hash Functions Properties

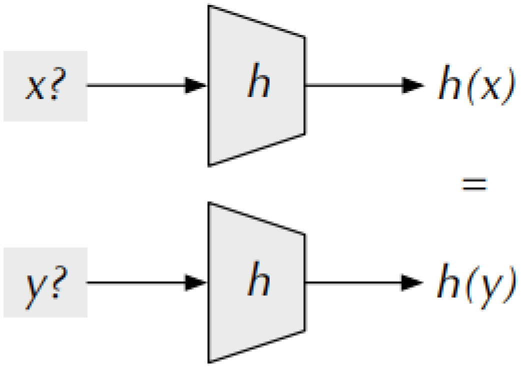
- **Preimage Resistance ("One-Way")**
  Given h(x) = z, hard to find x
  (or any input that hashes to z for that matter)



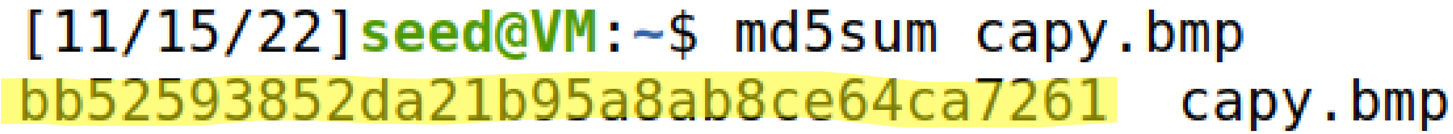


- **Second Preimage Resistance**
  Given x and h(x), hard to find y s.t. h(x) = h(y)

- 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 capy.bmp
bb52593852da21b95a8ab8ce64ca7261  capy.bmp
```

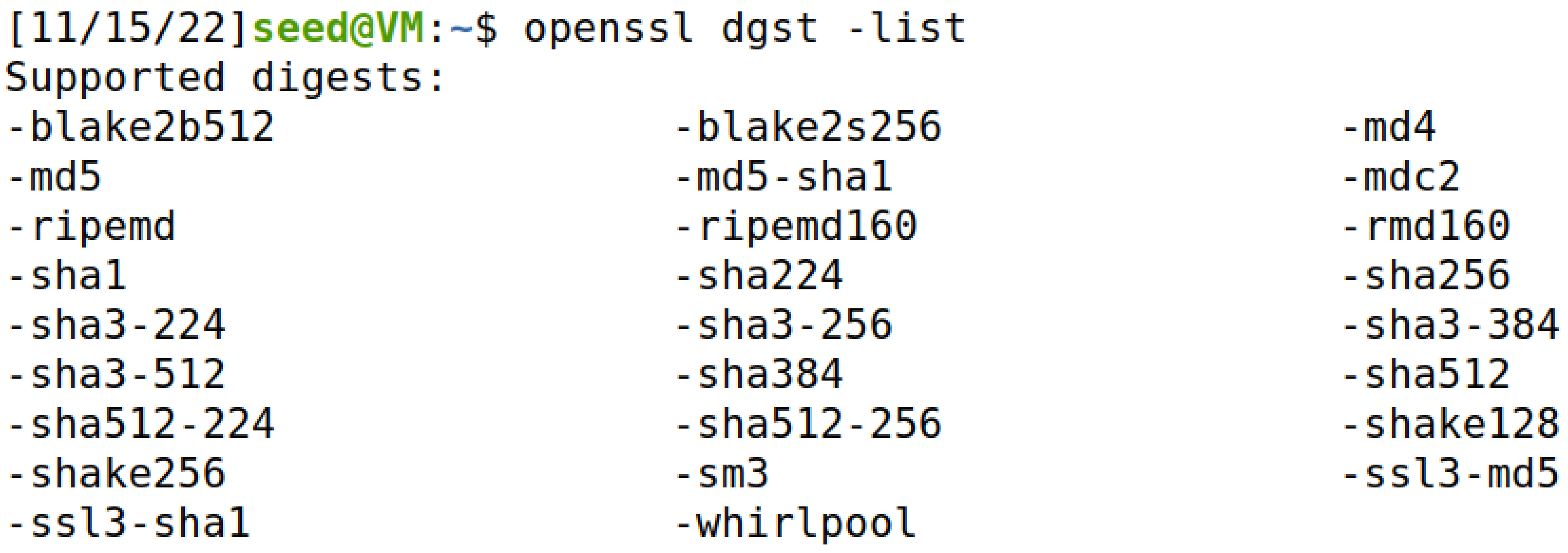
Gives an arbitrary size input a fixed-size unique* hash identifier

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

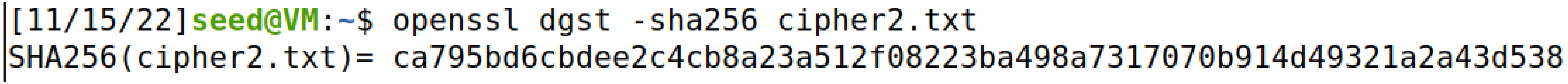
* We will break this

Computing Hashes with OpenSSL



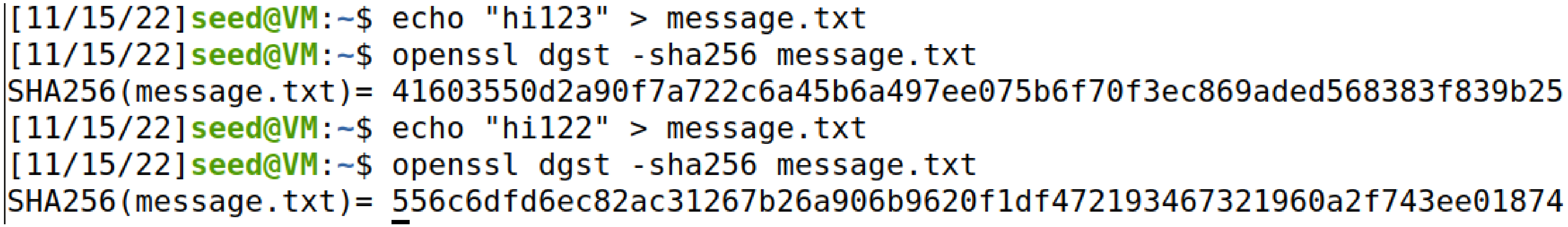
```
[11/15/22]seed@VM:~$ openssl dgst -list
Supported digests:
-blake2b512                 -blake2s256                 -md4
-md5                        -md5-sha1                   -mdc2
-ripemd                     -ripemd160                  -rmd160
-sha1                       -sha224                     -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



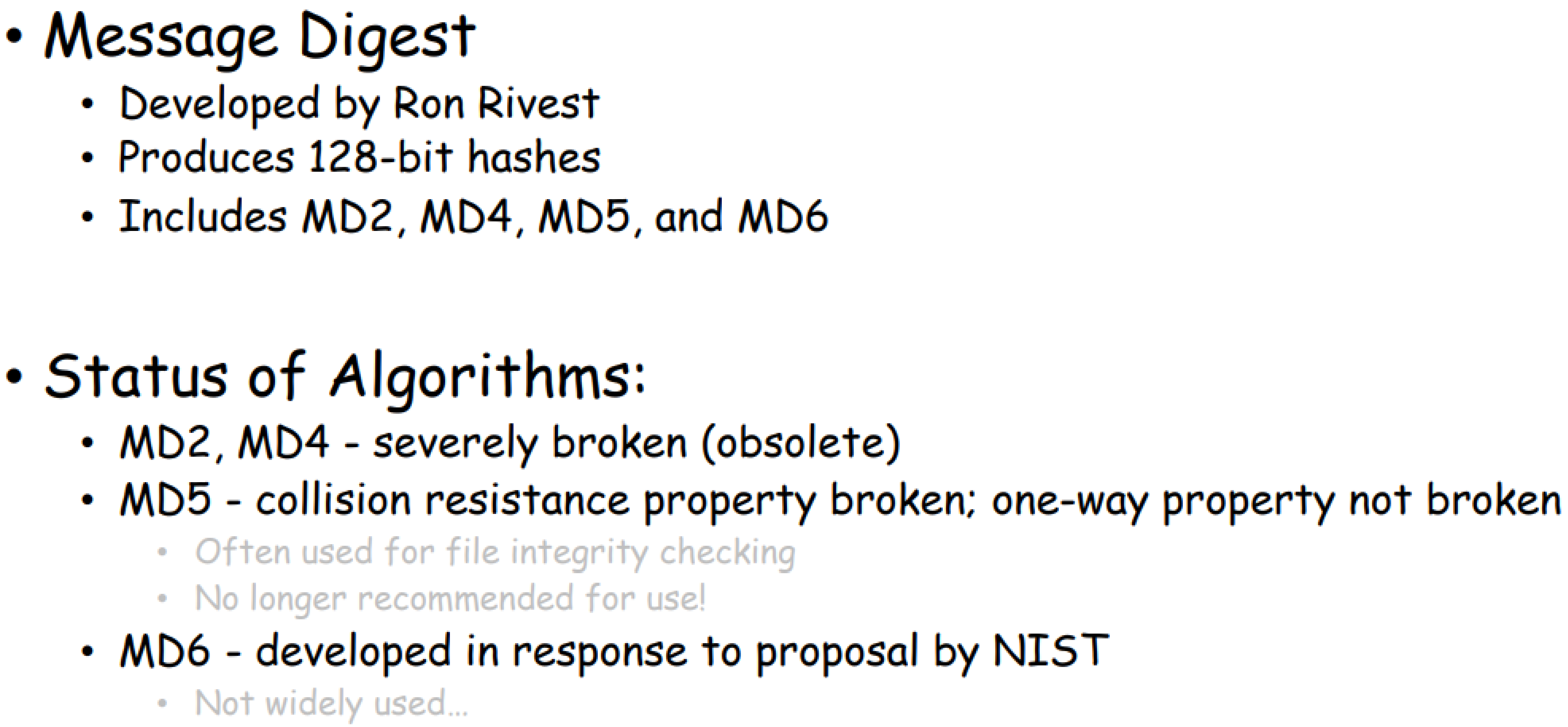
```
[11/15/22]seed@VM:~$ openssl dgst -sha256 cipher2.txt
SHA256(cipher2.txt)= ca795bd6cbdee2c4cb8a23a512f08223ba498a7317070b914d49321a2a43d538
```

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



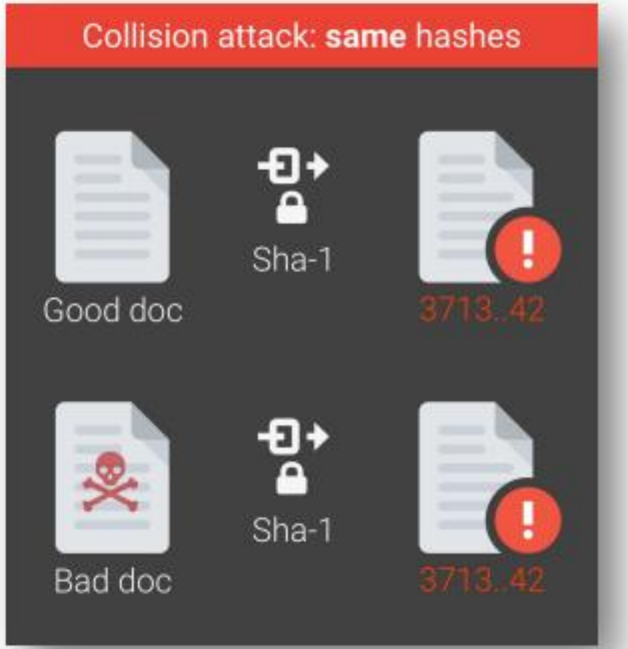
- Message Digest
  - Developed by Ron Rivest
  - Produces 128-bit hashes
  - Includes MD2, MD4, MD5, and MD6
- Status of Algorithms:
  - MD2, MD4 - severely broken (obsolete)
  - MD5 - collision resistance property broken; one-way property not broken
    - Often used for file integrity checking
    - 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

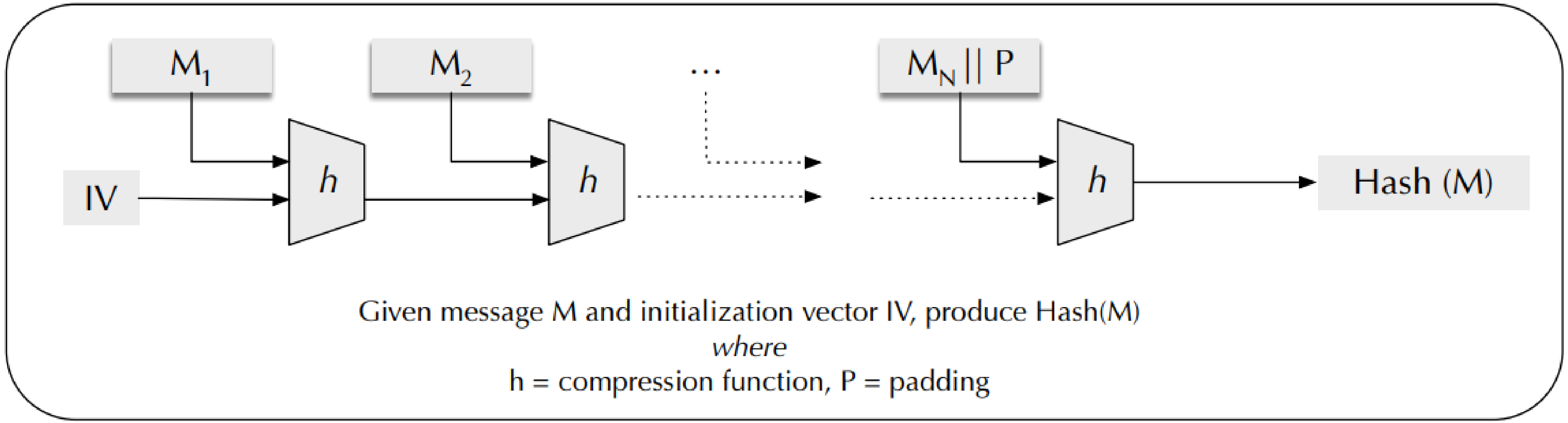
- Status of Algorithms:
  - SHA-0: withdrawn due to flaw
  - SHA-1: Designed by NSA Collision attack found in 2017
  - SHA-2: Designed by NSA
    - Includes SHA-256 and SHA-512 + other truncated versions;
    - No significant attack found yet…
  - SHA-3: Not Designed by NSA
    - Released in 2015; not 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?

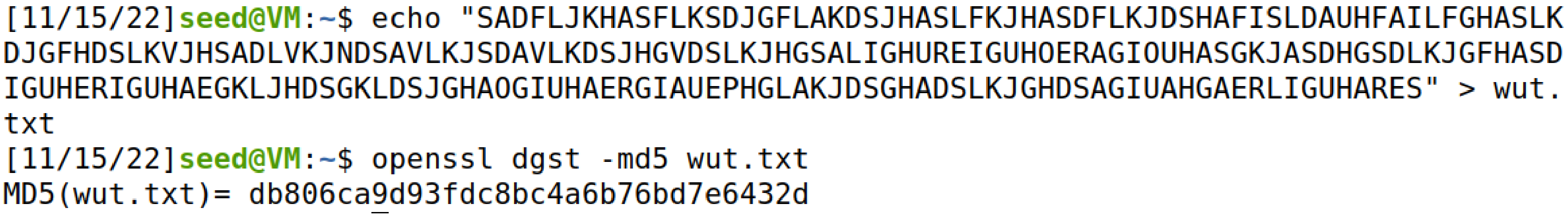
Most hash algorithms (e.g., MD5, SHA-1, SHA-2) use a Merkle-Damgard construction:



Given message M and initialization vector IV, produce Hash(M)
where
h = compression function, P = padding

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 "SADFLJKHASFLKSDJGFLAKDSJHASLFKJHASDFLKJDSHAFISLDAUHFAILFGHASLK
DJGFHDSLKVJHSADLVKJNDSAVLKJSDAVLKDSJHGVDSLKJHGSALIGHUREIGUHOERAGIOUHASGKJASDHGSDLKJGFHASD
IGUHERIGUHAEGKLJHDSGKLDSJGHAOGIUHAERGIAUEPHGLAKJDSGHADSLKJGHDSAGIUAHGAERLIGUHARES" > 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

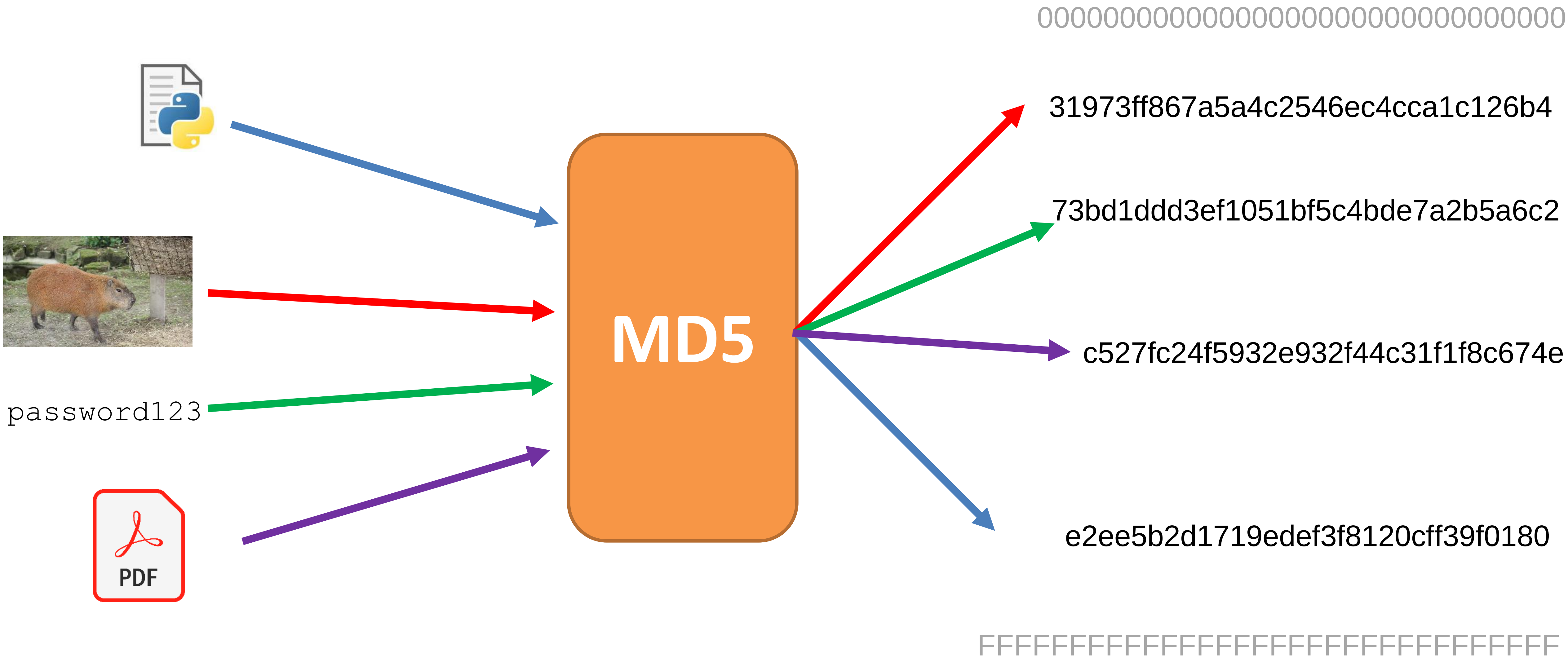
Calculating Hashes in Programming Languages

```
# 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

Output space of MD5 (128 bits)



00000000000000000000000000000000

password123

MD5

31973ff867a5a4c2546ec4cca1c126b4

73bd1ddd3ef1051bf5c4bde7a2b5a6c2

c527fc24f5932e932f44c31f1f8c674e

e2ee5b2d1719edef3f8120cff39f0180

FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

What are some uses for hashing?

Applications of Hashing

Integrity Verification

A CSCI 112 Student

`hello_world`

Applications of Hashing

Integrity Verification

A CSCI 112 Student

```
hello_world
```

Sent to professor for grading

```
hello_world
```

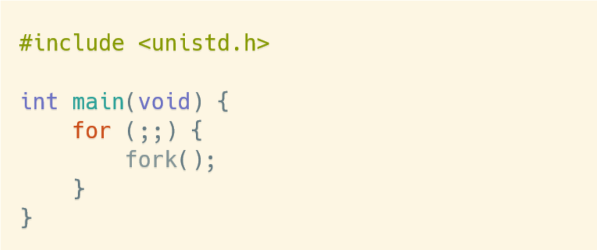
Instructor

Applications of Hashing

Integrity Verification

A CSCI 112 Student

hello_world



```
#include <unistd.h>

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

hello_world

What if the message got tampered with?

Instructor

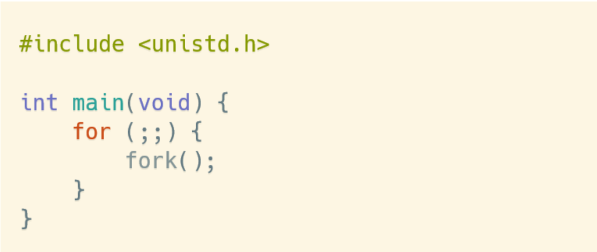
hello_world

Applications of Hashing

Integrity Verification

A CSCI 112 Student

hello_world



```
#include <unistd.h>

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

hello_world

What if the message got tampered with?

Instructor

hello_world

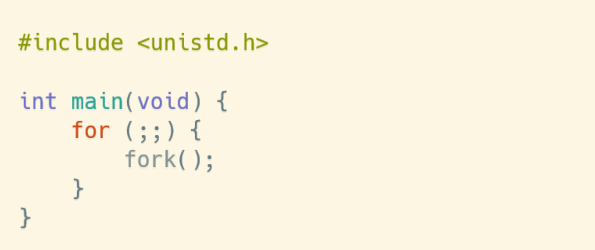
She will have no idea because this executable program seems totally normal and came from a trustworthy source

Applications of Hashing

Integrity Verification

A CSCI 112 Student

hello_world



```
#include <unistd.h>

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

hello_world

What if the message got tampered with?

Instructor

hello_world

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

Applications of Hashing

Integrity Verification

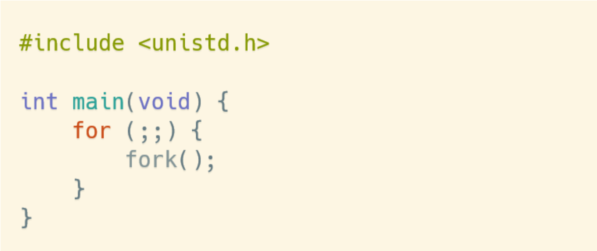
hello_world

89defae676abd3e3a42b41df17c40096

A CSCI 112 Student

Instructor

hello_world



```
#include <unistd.h>

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

hello_world

1. Generate hash for source file

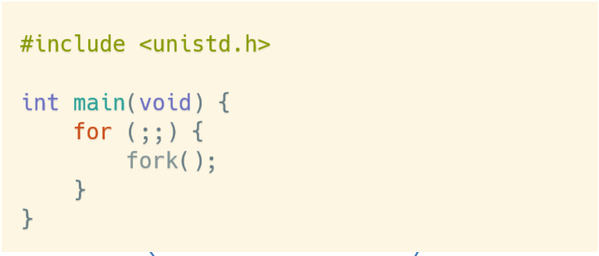
Applications of Hashing

Integrity Verification

hello_world

89defae676abd3e3a42b41df17c40096

A CSCI 112 Student



```
#include <unistd.h>

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

hello_world

hello_world

b0608c4e1775ad8f92e7b5c191774c5d

Instructor

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

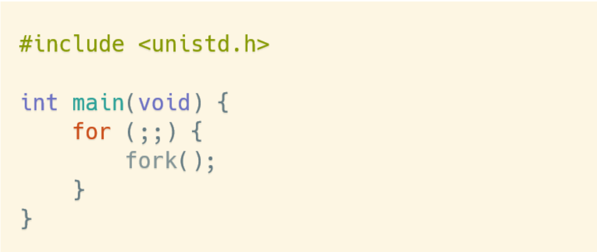
Applications of Hashing

Integrity Verification

A CSCI 112 Student

hello_world

89defae676abd3e3a42b41df17c40096



```
#include <unistd.h>

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

hello_world

Instructor

hello_world

b0608c4e1775ad8f92e7b5c191774c5d

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

The hashes do not match!

Applications of Hashing

Integrity Verification

hello_world

89defae676abd3e3a42b41df17c40096

A CSCI 112 Student

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

Different hashes = Something fishy happened!

hello_world

b0608c4e1775ad8f92e7b5c191774c5d

Instructor

Applications of Hashing

Integrity Verification

A CSCI 112 Student

hello_world

89defae676abd3e3a42b41df17c40096

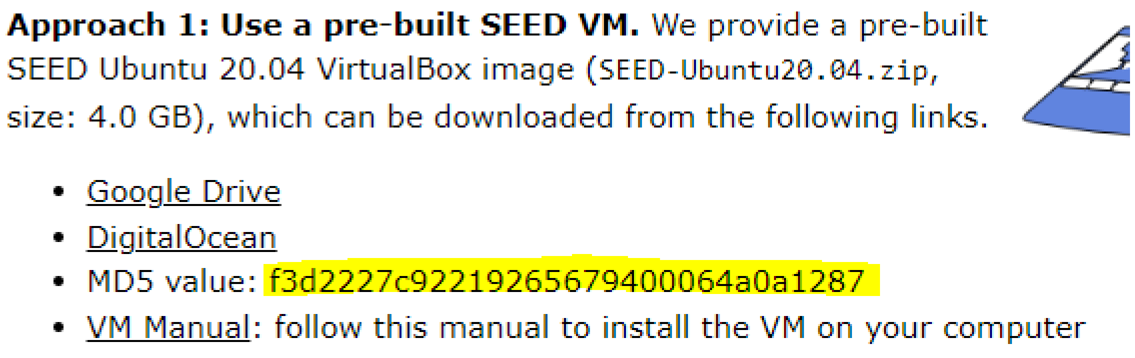
Instructor

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

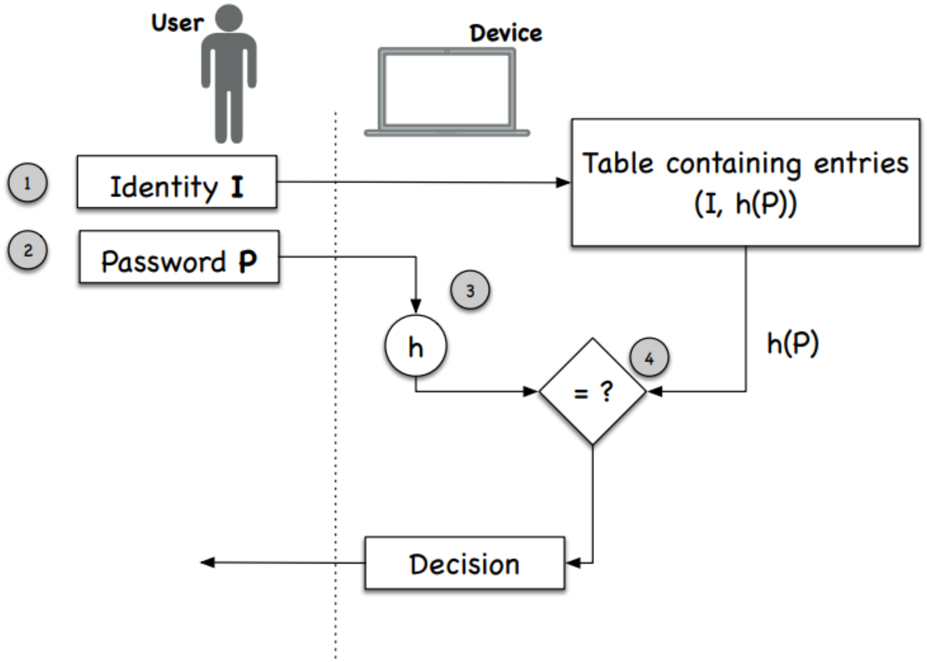
Applications of Hashing

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

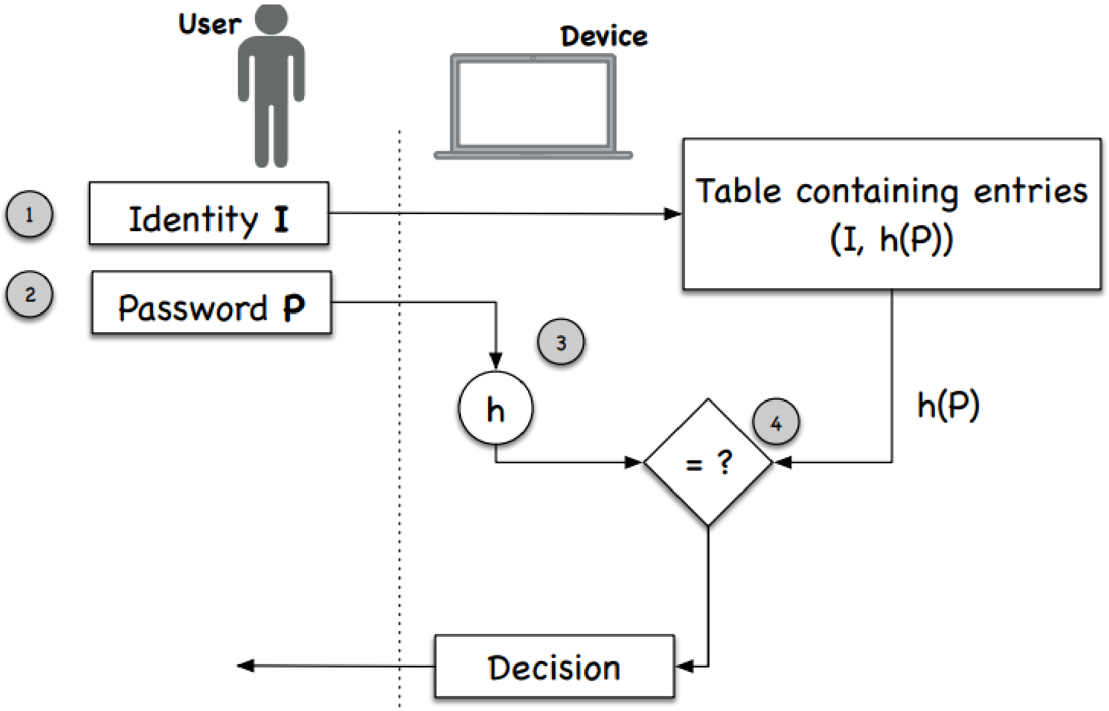


Password Verification

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



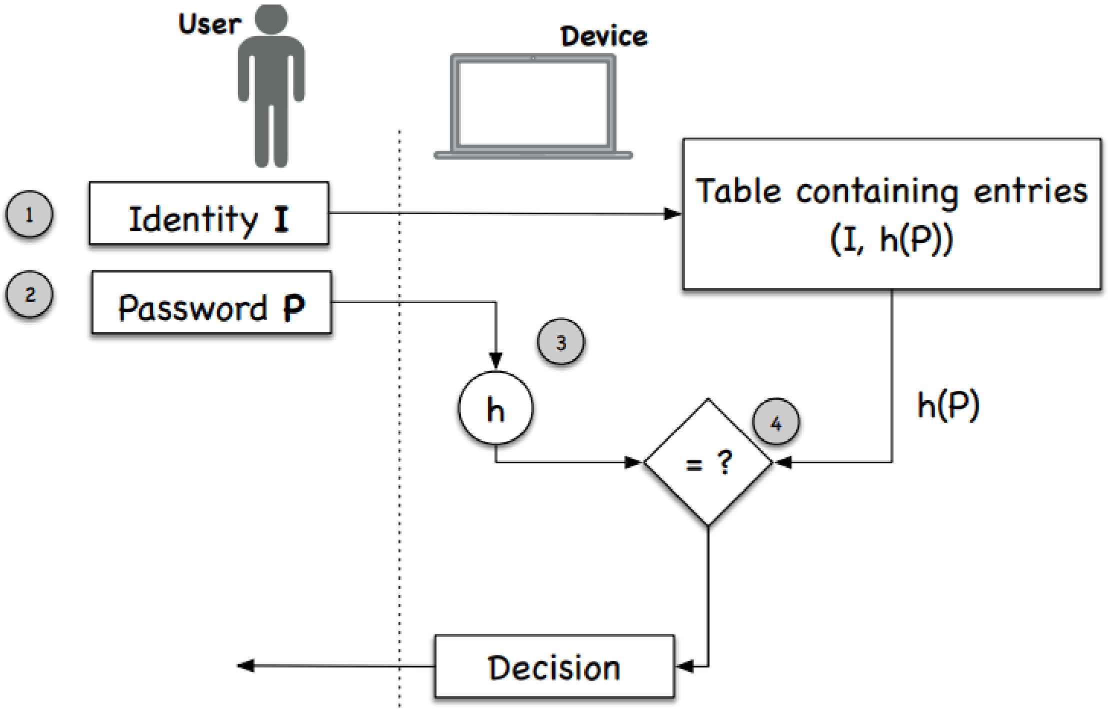
If two people have the same password….

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



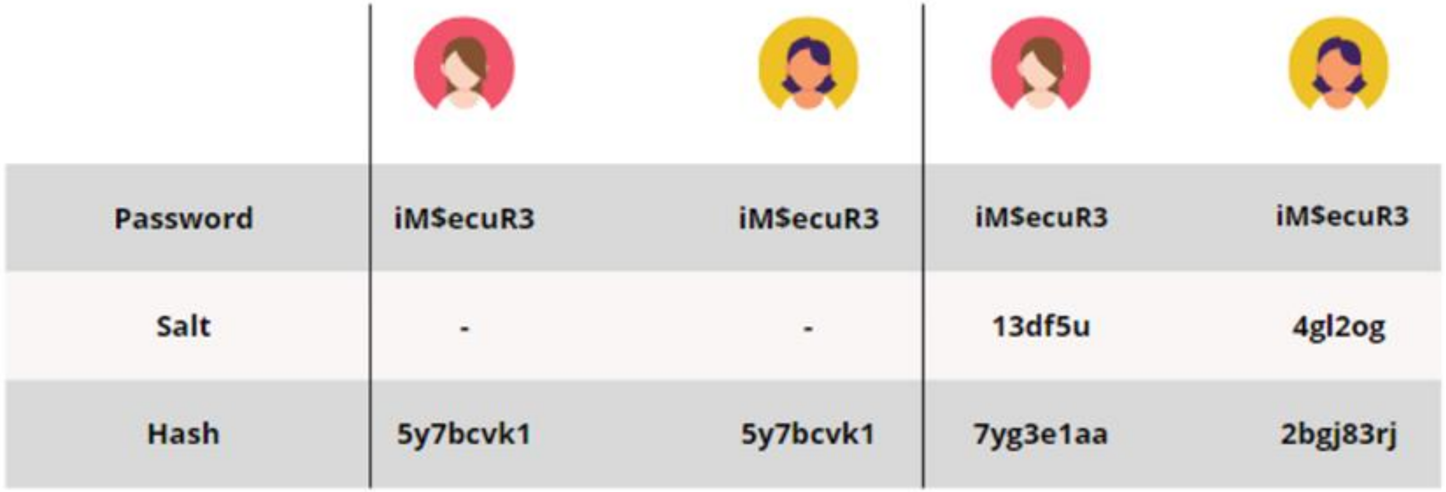
If two people have the same password….
Their passwords will be the same!

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!

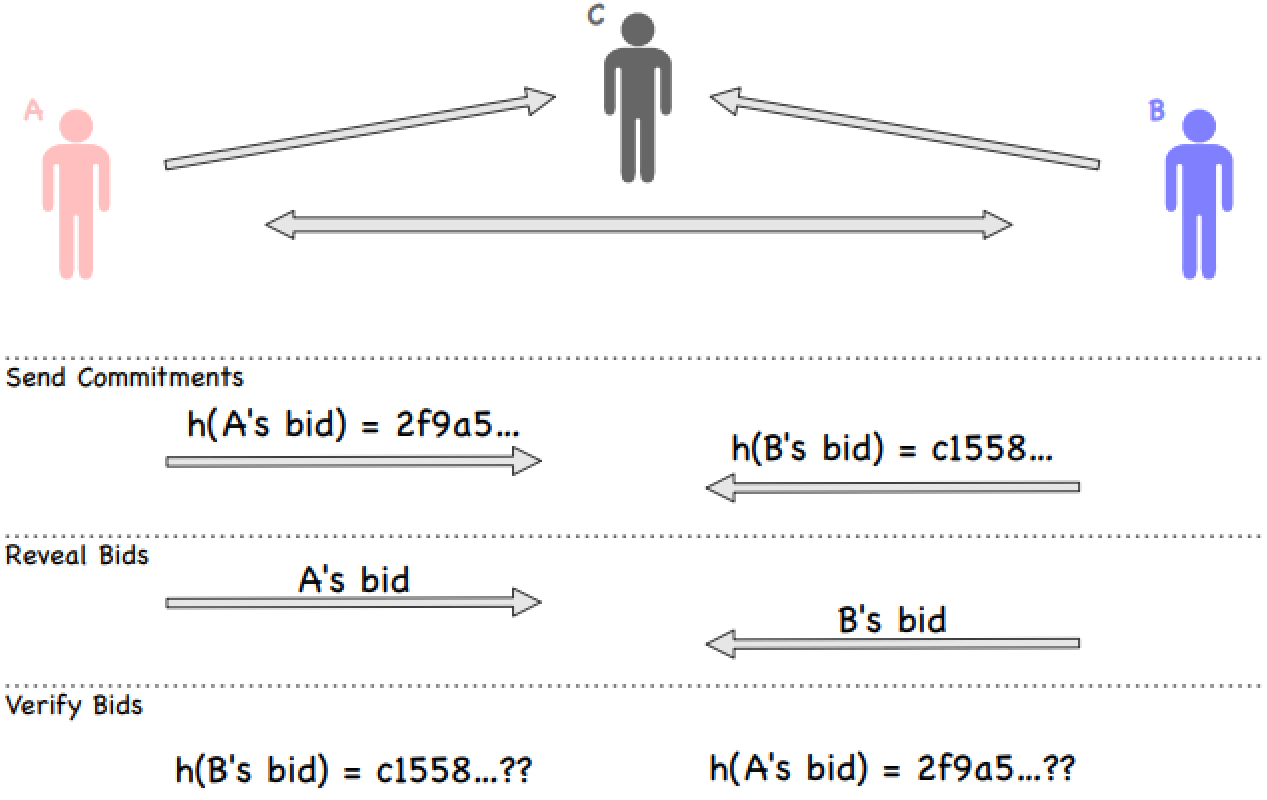


Password	iM$ecuR3	iM$ecuR3	iM$ecuR3	iM$ecuR3
Salt	-	-	13df5u	4gl2og
Hash	5y7bcvk1	5y7bcvk1	7yg3e1aa	2bgj83rj

Fairness and Commitment (scary)

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

- Example: Fair Games



Send Commitments

h(A's bid) = 2f9a5...

h(B's bid) = c1558...

Reveal Bids

A's bid

B's bid

Verify Bids

h(B's bid) = c1558...??

h(A's bid) = 2f9a5...??

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

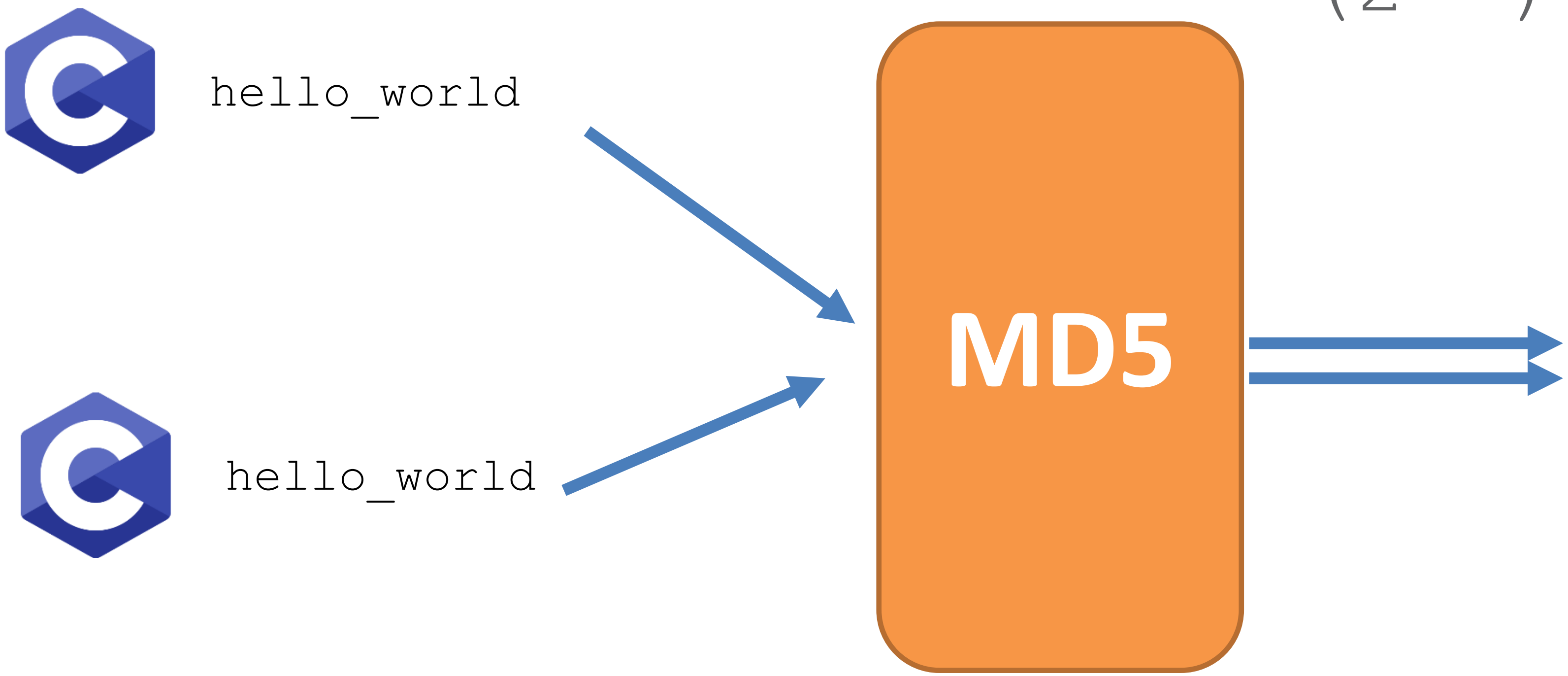
Hash Collision Attacks compromise the integrity of a program by creating a malicious file that has a same hash

Collision Attacks

00000000000000000000000000000000
00000000000000000000000000000001
00000000000000000000000000000010

There is a very large amount of possible hashes

~ (2^{128})



106a7d06be131315e25a7cbe57af398c

...

...

EEFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
EFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

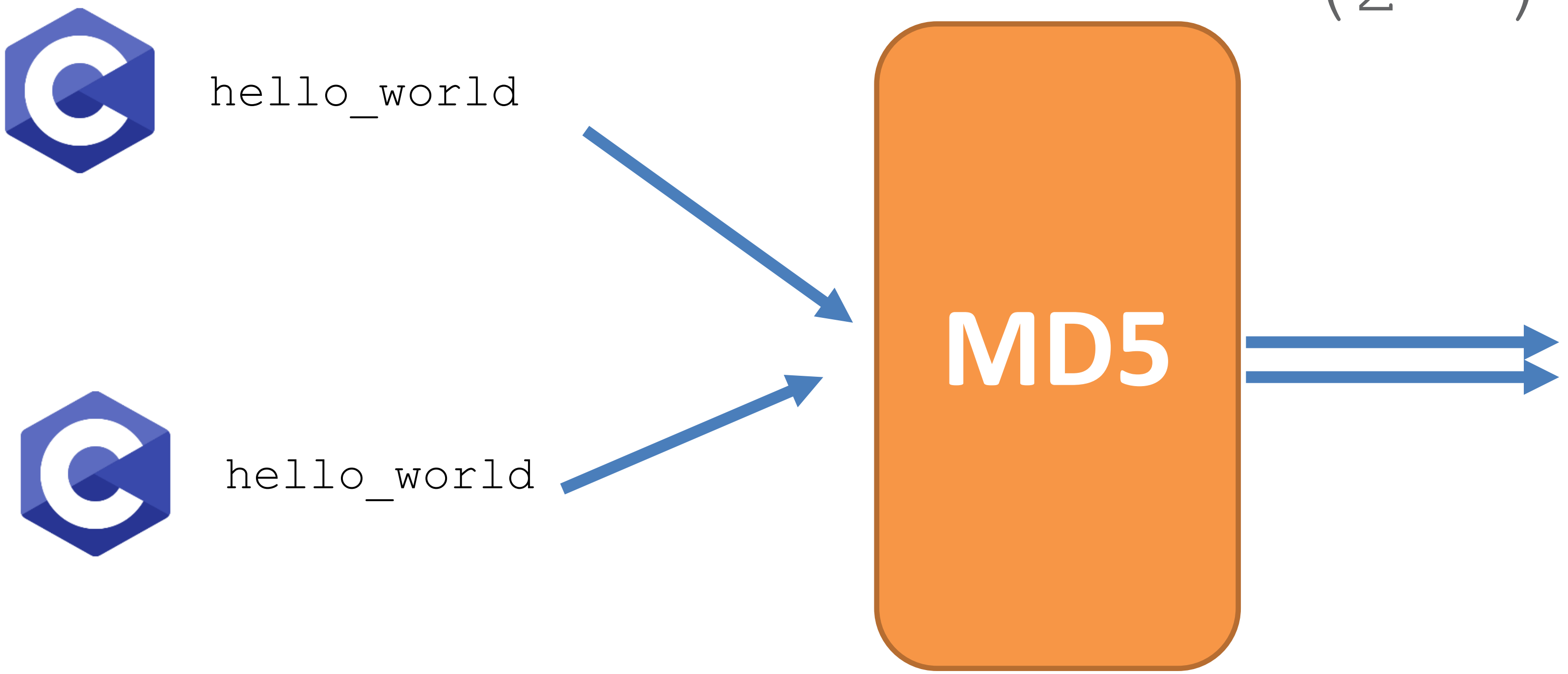
How likely is? Very unlikely?

Collision Attacks

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

00000000000000000000000000000000
00000000000000000000000000000001
00000000000000000000000000000010

…



hello_world

hello_world

MD5

106a7d06be131315e25a7cbe57af398c

…

How likely is? Very unlikely?

More likely than you think…

EEFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
EFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

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

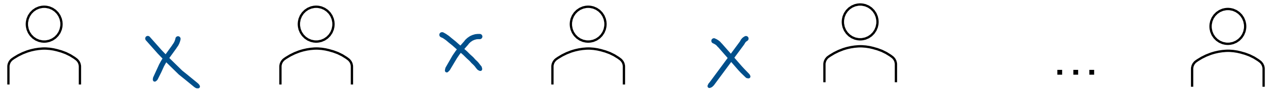
365/365 × 364/365 × 363/365 × 362/365 … 343/365

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 × 362/365 … 343/365

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

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

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

$1 - .4927$

$= \approx 50\%$

Go Watch this video: https://www.youtube.com/watch?v=ofTb57aZHZs

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…

Go Watch this video: https://www.youtube.com/watch?v=ofTb57aZHZs

Announcements

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

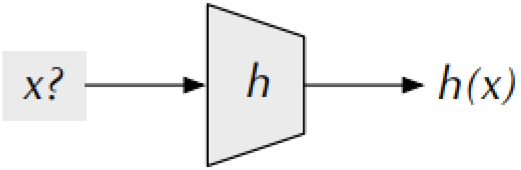
Email me if you need anything over the break

Last day to drop with a W is today

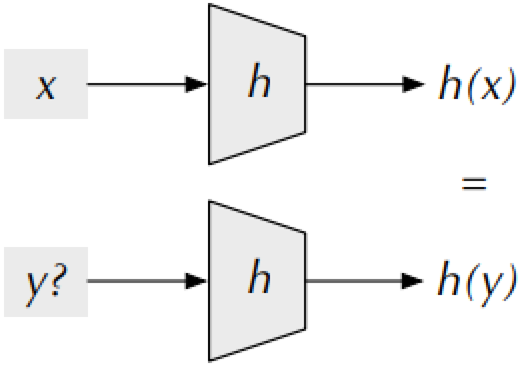
XOR task

Hash Functions Properties

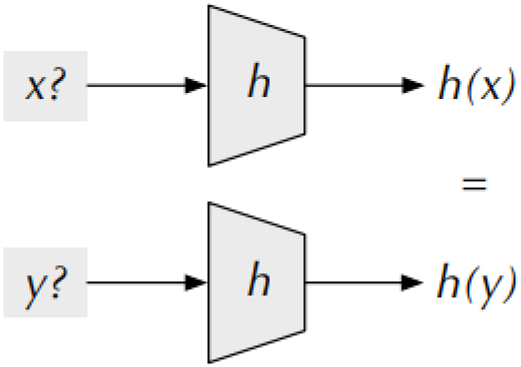
- **Preimage Resistance ("One-Way")**
  Given h(x) = z, hard to find x
  (or any input that hashes to z for that matter)



- **Second Preimage Resistance**
  Given x and h(x), hard to find y s.t. h(x) = h(y)



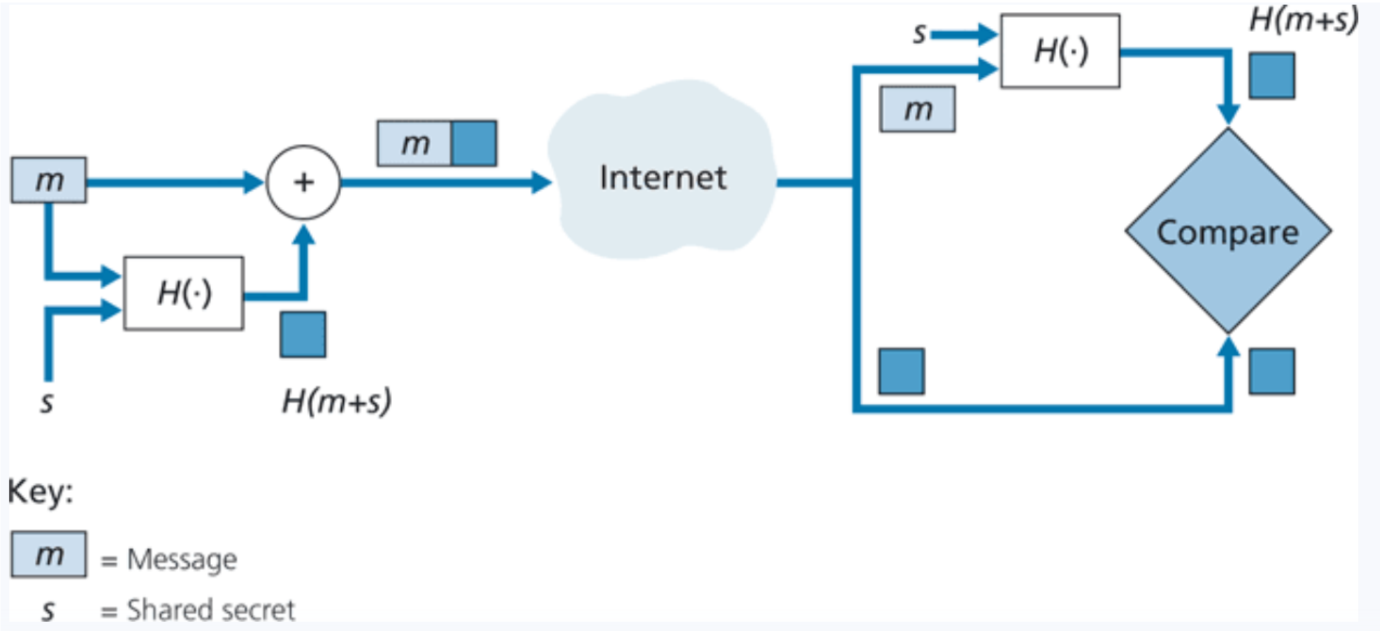
- Collision Resistance (or, ideally, "Collision Free")
  Difficult to find x and y s.t. hash(x) = hash(y)



Uses for Hashing: Message Authentication Code (MAC)

1. Append a message with a shared secret (m + s)
2. Compute hash of (m+s) → 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?

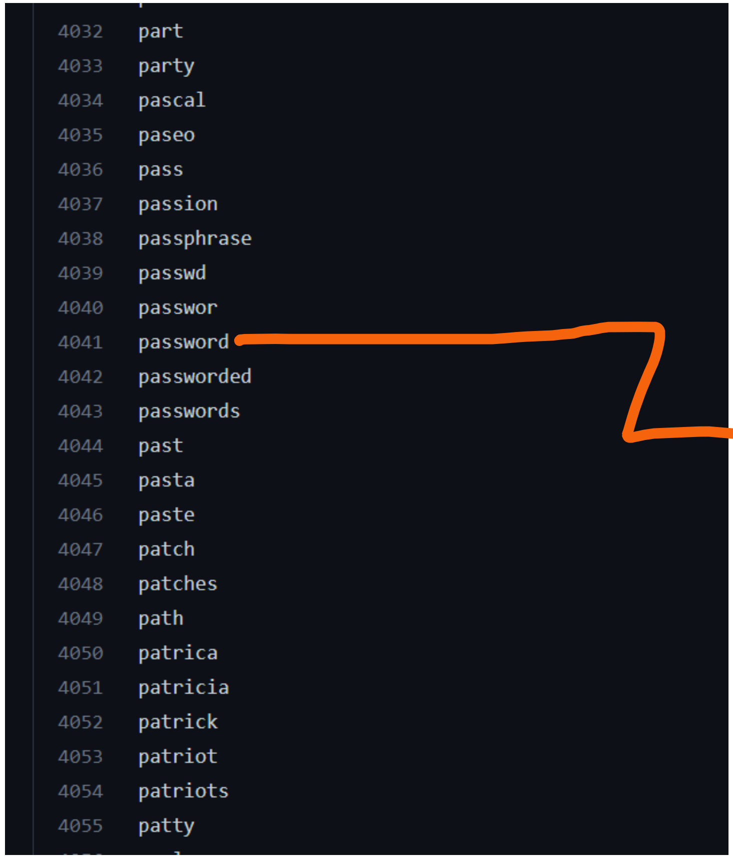
```
afc285bebb3dd733796cb06db01cd59a
```

Techniques

- Dictionary Attack
- Rainbow Tables

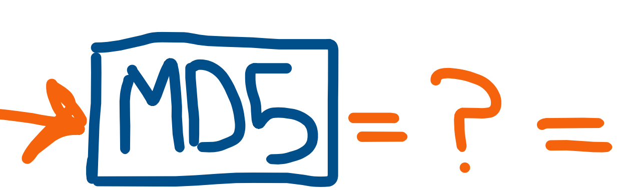
Dictionary Attack

We will use an existing list of common passwords



```
part
party
pascal
paseo
pass
passion
passphrase
passwd
passwor
password
passworded
passwords
past
pasta
paste
patch
patches
path
patrica
patricia
patrick
patriot
patriots
patty
```

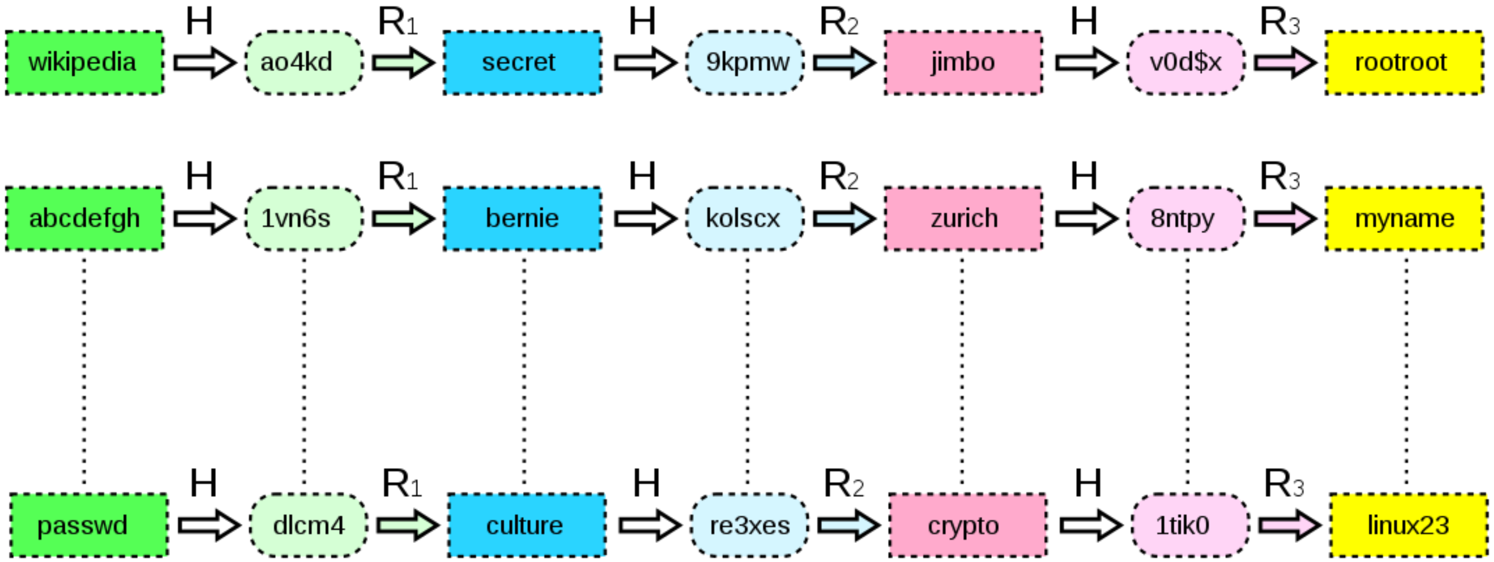
1. Iterate through each line of file
2. Compute hash of word
3. Check for match



afc285bebb3dd733796cb06db01cd59a

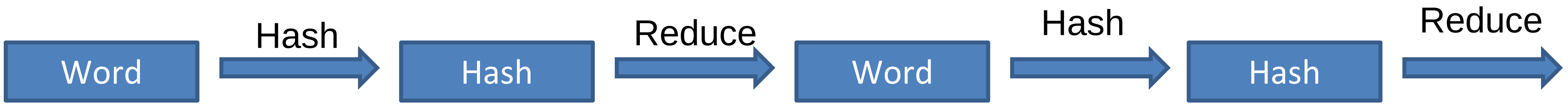
This works for cracking weak, unsalted passwords

Rainbow Tables



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

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

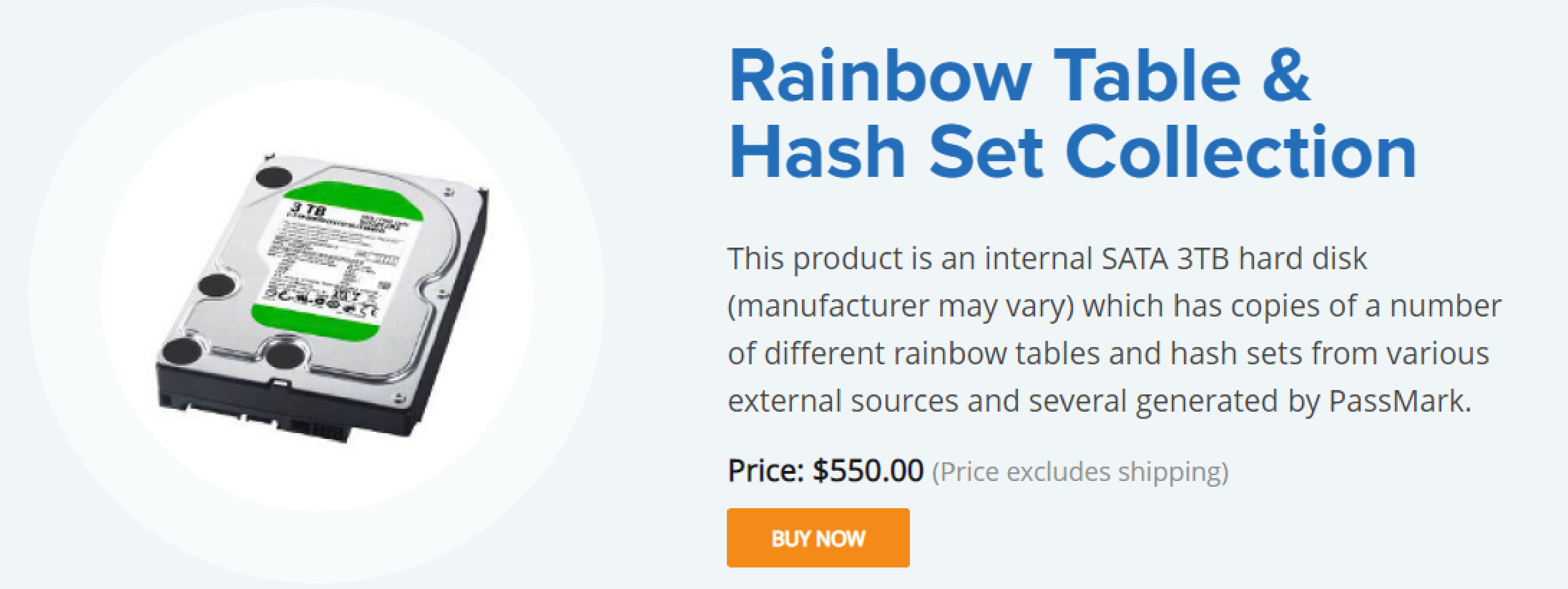


Looking up a value in the rainbow table can happen quick, but these files are typically very large

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.

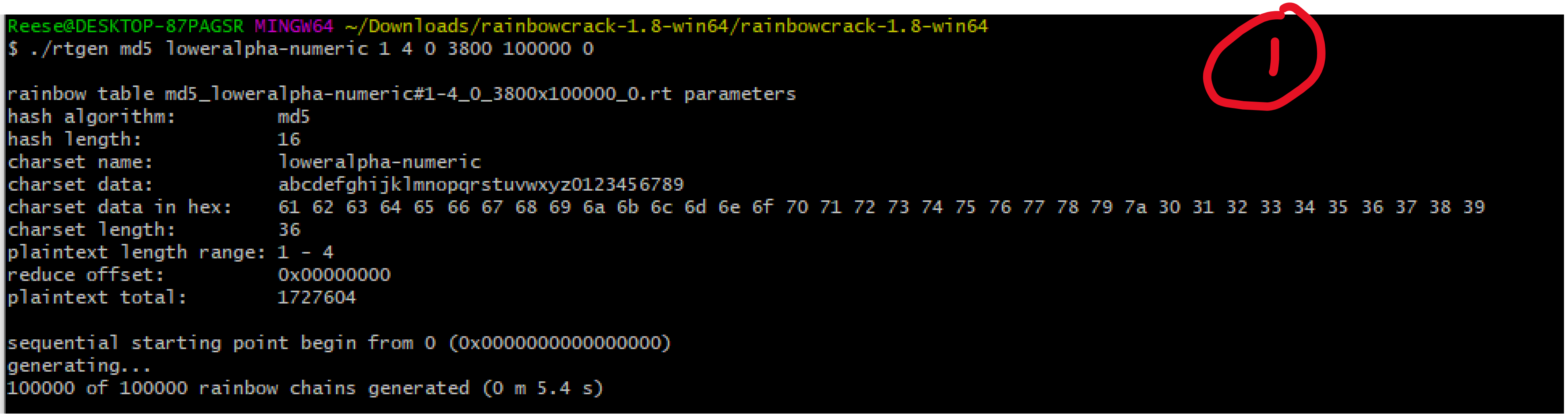
Price: $550.00 (Price excludes shipping)

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

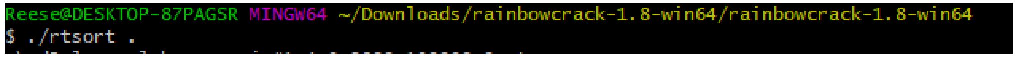
Rainbow Tables using RainbowCrack



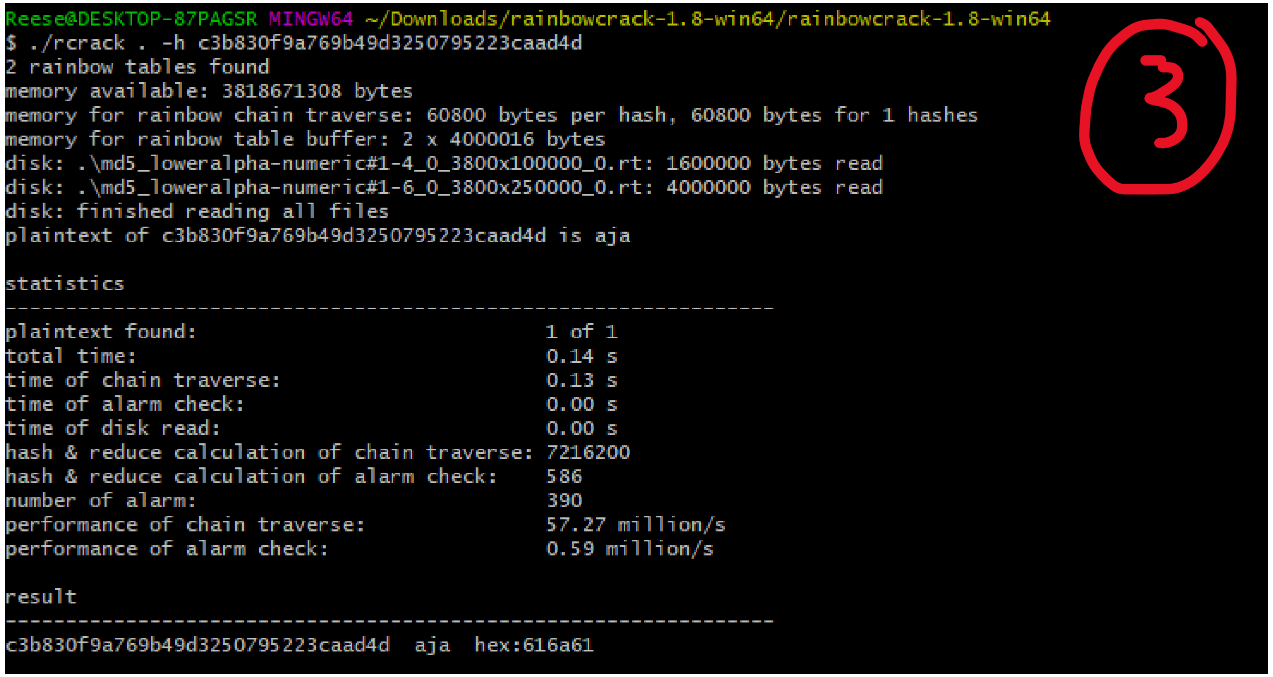
1

```
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 .
```



3

```
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
```

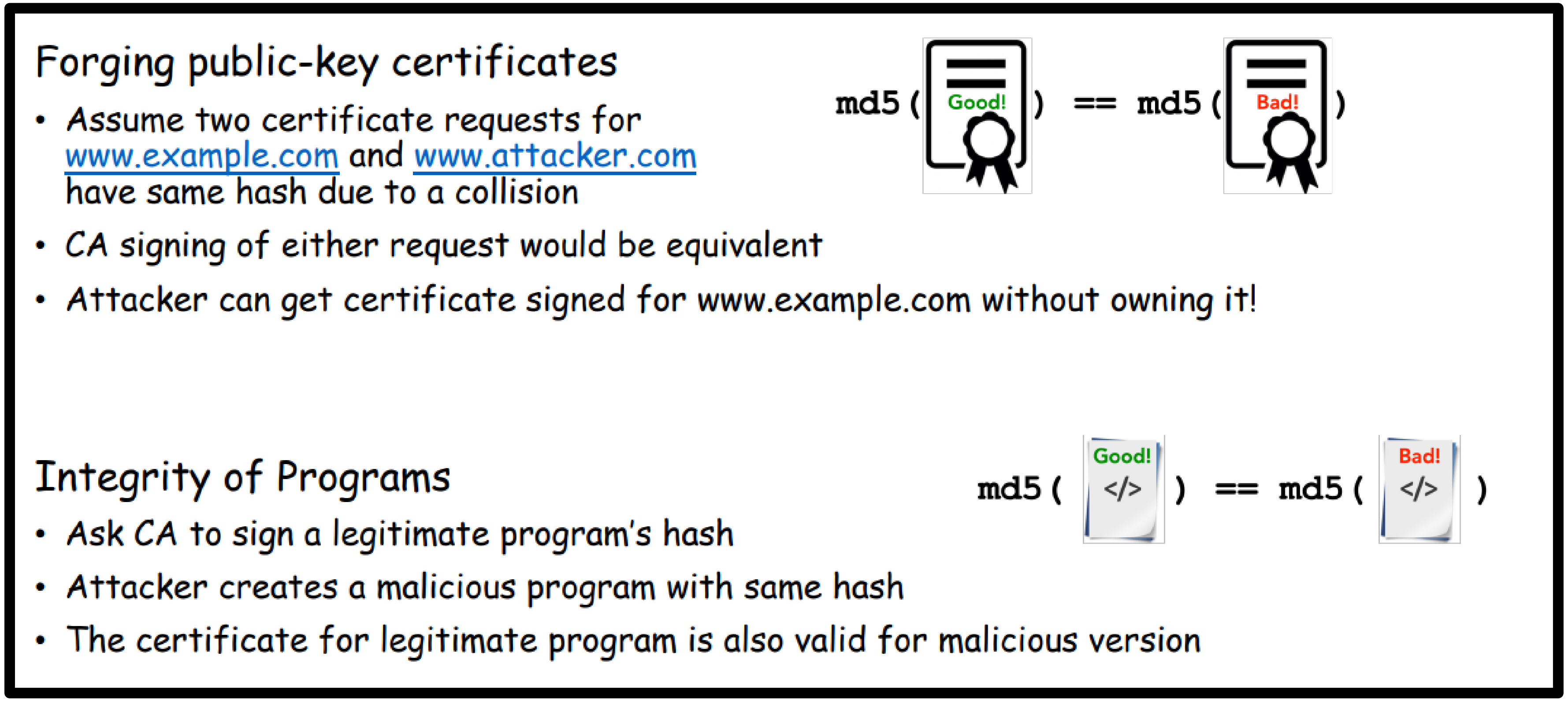
Hash Collisions

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!

md5 (Good!) == md5 (Bad!)

Integrity of Programs

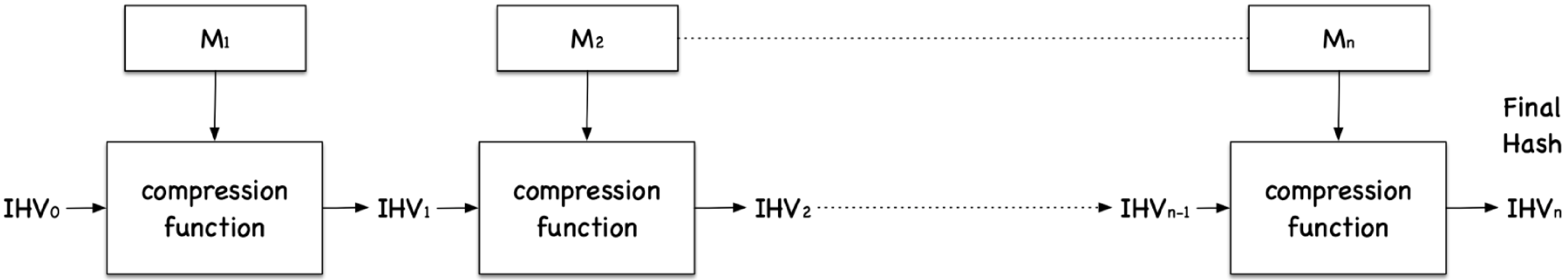
- Ask CA to sign a legitimate program's hash
- Attacker creates a malicious program with same hash
- The certificate for legitimate program is also valid for malicious version

md5 (Good! </>) == md5 (Bad! </>)

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!

MD5 Under the hood



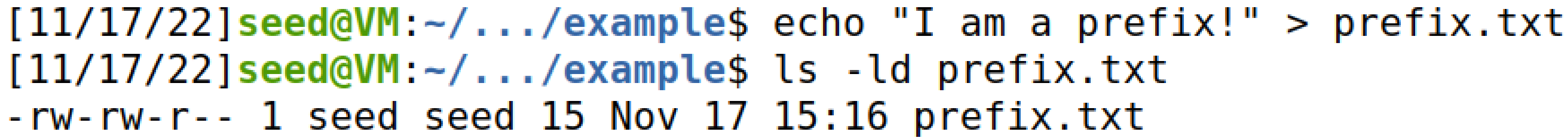
Fact: Message is divided into blocks, and each block is run through a compression function

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!

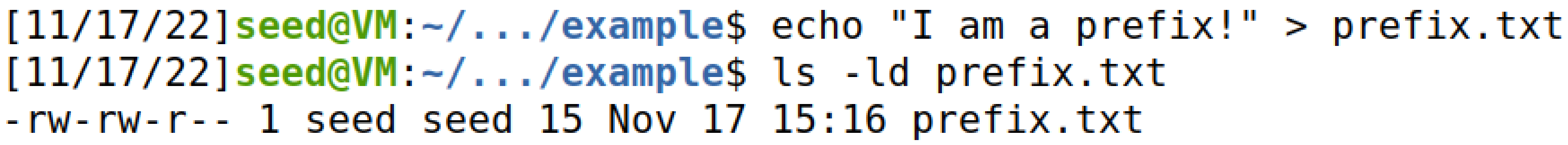


```
[11/17/22]seed@VM:~/.../example$ echo "I am a prefix!" > prefix.txt
[11/17/22]seed@VM:~/.../example$ ls -ld prefix.txt
-rw-rw-r-- 1 seed seed 15 Nov 17 15:16 prefix.txt
```

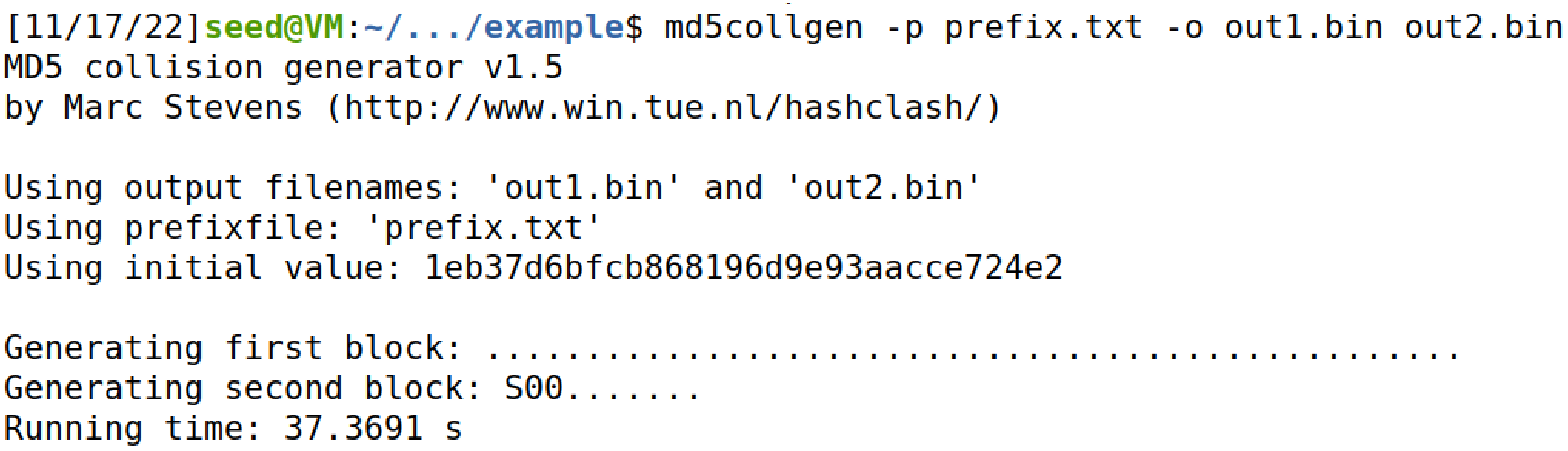
Hash Collisions (MD5collgen)

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
```

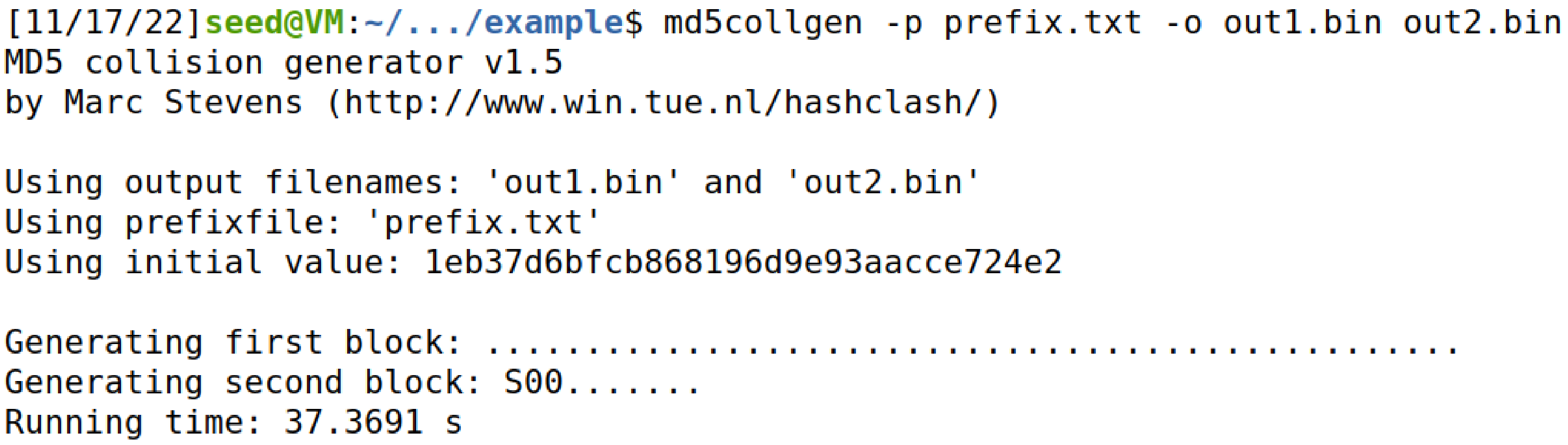


```
[11/17/22]seed@VM:~/.../example$ md5collgen -p prefix.txt -o out1.bin out2.bin
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

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

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

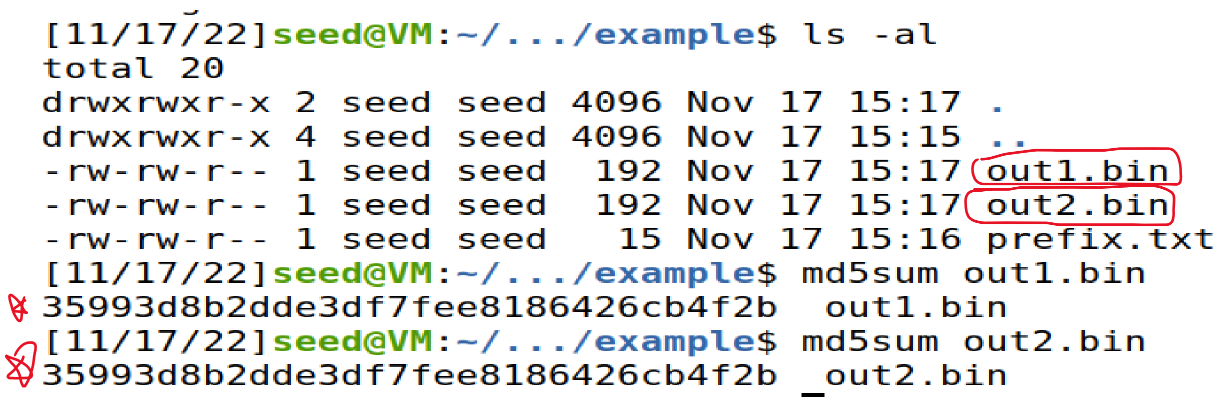
Hash Collisions (MD5collgen)

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
```



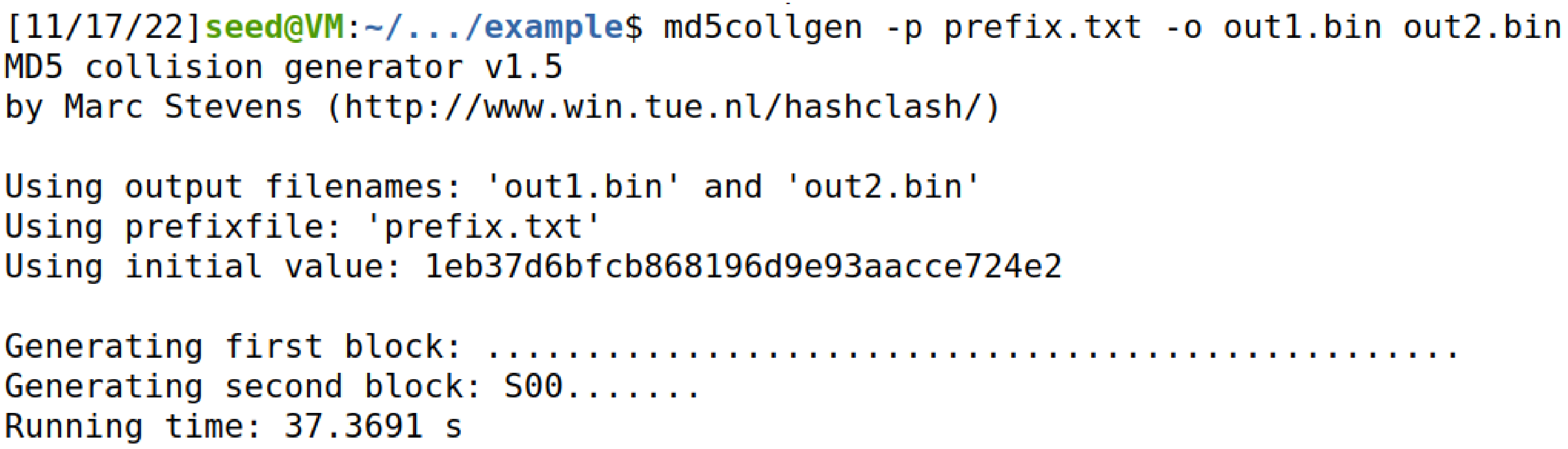
```
[11/17/22]seed@VM:~/.../example$ ls -al
total 20
drwxrwxr-x 2 seed seed 4096 Nov 17 15:17 .
drwxrwxr-x 4 seed seed 4096 Nov 17 15:15 ..
-rw-rw-r-- 1 seed seed  192 Nov 17 15:17 out1.bin
-rw-rw-r-- 1 seed seed  192 Nov 17 15:17 out2.bin
-rw-rw-r-- 1 seed seed   15 Nov 17 15:16 prefix.txt
[11/17/22]seed@VM:~/.../example$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

Same Hash!

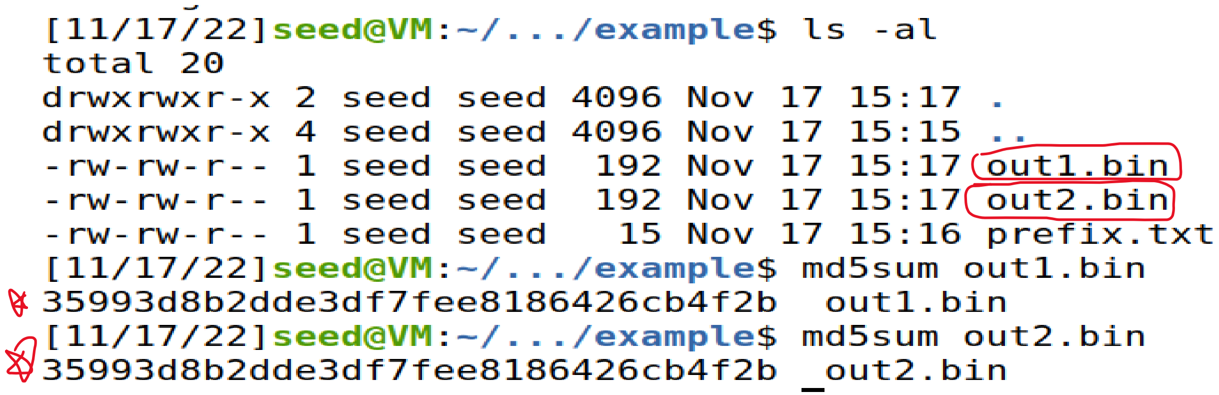
Hash Collisions (MD5collgen)

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
```



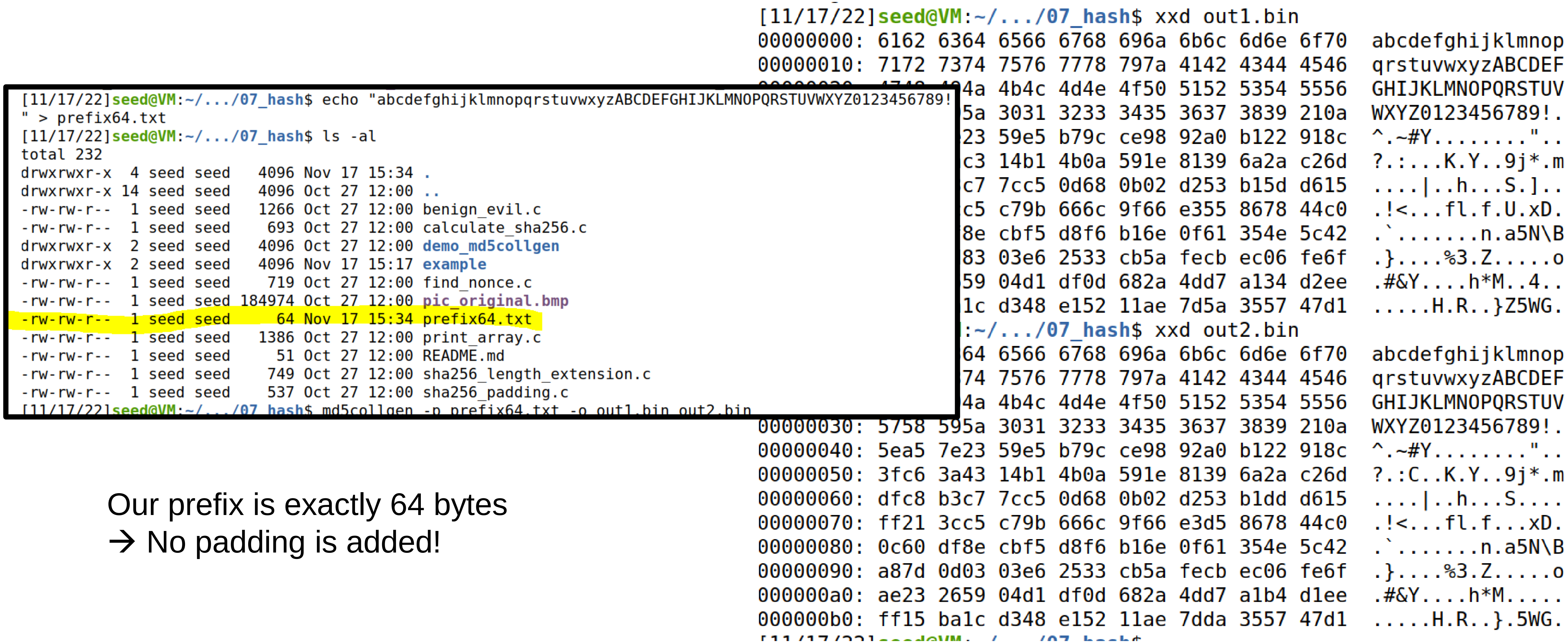
```
[11/17/22]seed@VM:~/.../example$ ls -al
total 20
drwxrwxr-x 2 seed seed 4096 Nov 17 15:17 .
drwxrwxr-x 4 seed seed 4096 Nov 17 15:15 ..
-rw-rw-r-- 1 seed seed  192 Nov 17 15:17 out1.bin
-rw-rw-r-- 1 seed seed  192 Nov 17 15:17 out2.bin
-rw-rw-r-- 1 seed seed   15 Nov 17 15:16 prefix.txt
[11/17/22]seed@VM:~/.../example$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



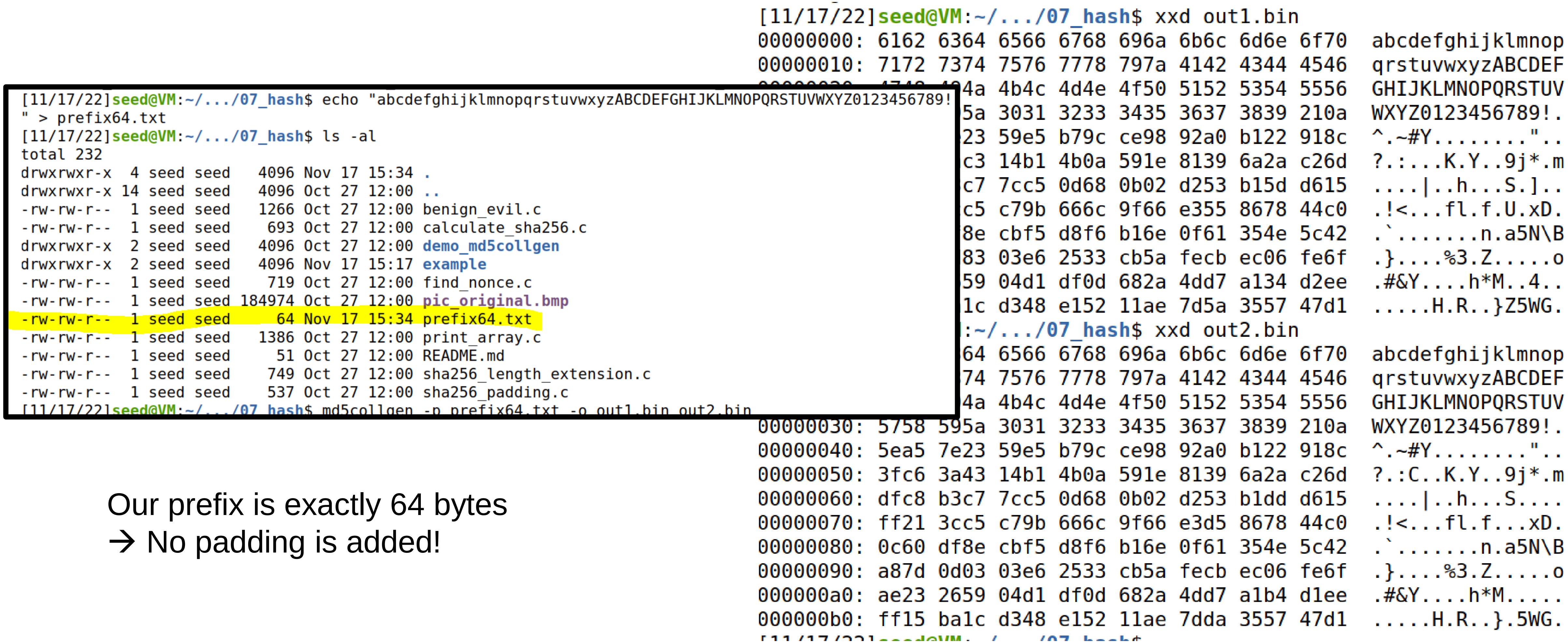
```
[11/17/22]seed@VM:~/.../07_hash$ xxd out1.bin
00000000: 6162 6364 6566 6768 696a 6b6c 6d6e 6f70  abcdefghijklmnop
00000010: 7172 7374 7576 7778 797a 4142 4344 4546  qrstuvwxyzABCDEF
              4a 4b4c 4d4e 4f50 5152 5354 5556  GHIJKLMNOPQRSTUV
              5a 3031 3233 3435 3637 3839 210a  WXYZ0123456789!.
              23 59e5 b79c ce98 92a0 b122 918c  ^.~#Y........"..
              c3 14b1 4b0a 591e 8139 6a2a c26d  ?.:...K.Y..9j*.m
              c7 7cc5 0d68 0b02 d253 b15d d615  ....|..h...S.]..
              c5 c79b 666c 9f66 e355 8678 44c0  .!<...fl.f.U.xD.
              8e cbf5 d8f6 b16e 0f61 354e 5c42  .`.......n.a5N\B
              83 03e6 2533 cb5a fecb ec06 fe6f  .}....%3.Z.....o
              59 04d1 df0d 682a 4dd7 a134 d2ee  .#&Y....h*M..4..
              1c d348 e152 11ae 7d5a 3557 47d1  .....H.R..}Z5WG.
            :~/.../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
00000030: 5758 595a 3031 3233 3435 3637 3839 210a  WXYZ0123456789!.
00000040: 5ea5 7e23 59e5 b79c ce98 92a0 b122 918c  ^.~#Y........"..
00000050: 3fc6 3a43 14b1 4b0a 591e 8139 6a2a c26d  ?.:C..K.Y..9j*.m
00000060: dfc8 b3c7 7cc5 0d68 0b02 d253 b1dd d615  ....|..h...S....
00000070: ff21 3cc5 c79b 666c 9f66 e3d5 8678 44c0  .!<...fl.f...xD.
00000080: 0c60 df8e cbf5 d8f6 b16e 0f61 354e 5c42  .`.......n.a5N\B
00000090: a87d 0d03 03e6 2533 cb5a fecb ec06 fe6f  .}....%3.Z.....o
000000a0: ae23 2659 04d1 df0d 682a 4dd7 a1b4 d1ee  .#&Y....h*M.....
000000b0: ff15 ba1c d348 e152 11ae 7dda 3557 47d1  .....H.R..}.5WG.
[11/17/22]seed@VM:~/.../07_hash$
```

```
[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!

Hash Collisions (MD5collgen)

What if out prefix is a multiple of 64?



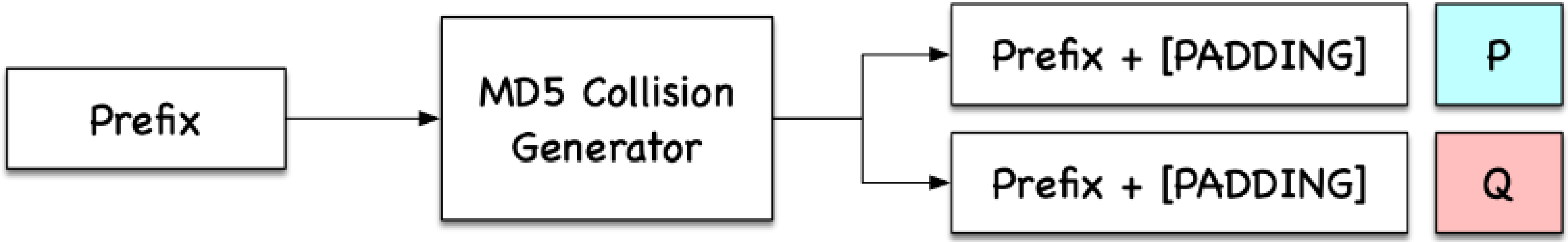
```
[11/17/22]seed@VM:~/.../07_hash$ echo "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!
" > prefix64.txt
[11/17/22]seed@VM:~/.../07_hash$ ls -al
total 232
drwxrwxr-x  4 seed seed   4096 Nov 17 15:34 .
drwxrwxr-x 14 seed seed   4096 Oct 27 12:00 ..
-rw-rw-r--  1 seed seed   1266 Oct 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
```

```
[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
                    4a 4b4c 4d4e 4f50 5152 5354 5556  GHIJKLMNOPQRSTUV
                    5a 3031 3233 3435 3637 3839 210a  WXYZ0123456789!.
                    23 59e5 b79c ce98 92a0 b122 918c  ^.~#Y........"..
                    c3 14b1 4b0a 591e 8139 6a2a c26d  ?.:...K.Y..9j*.m
                    c7 7cc5 0d68 0b02 d253 b15d d615  ....|..h...S.]..
                    c5 c79b 666c 9f66 e355 8678 44c0  .!<...fl.f.U.xD.
                    8e cbf5 d8f6 b16e 0f61 354e 5c42  .`.......n.a5N\B
                    83 03e6 2533 cb5a fecb ec06 fe6f  .}....%3.Z.....o
                    59 04d1 df0d 682a 4dd7 a134 d2ee  .#&Y....h*M..4..
                    1c d348 e152 11ae 7d5a 3557 47d1  .....H.R..}Z5WG.
                    :~/.../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
00000030: 5758 595a 3031 3233 3435 3637 3839 210a  WXYZ0123456789!.
00000040: 5ea5 7e23 59e5 b79c ce98 92a0 b122 918c  ^.~#Y........"..
00000050: 3fc6 3a43 14b1 4b0a 591e 8139 6a2a c26d  ?.:C..K.Y..9j*.m
00000060: dfc8 b3c7 7cc5 0d68 0b02 d253 b1dd d615  ....|..h...S....
00000070: ff21 3cc5 c79b 666c 9f66 e3d5 8678 44c0  .!<...fl.f...xD.
00000080: 0c60 df8e cbf5 d8f6 b16e 0f61 354e 5c42  .`.......n.a5N\B
00000090: a87d 0d03 03e6 2533 cb5a fecb ec06 fe6f  .}....%3.Z.....o
000000a0: ae23 2659 04d1 df0d 682a 4dd7 a1b4 d1ee  .#&Y....h*M.....
000000b0: ff15 ba1c d348 e152 11ae 7dda 3557 47d1  .....H.R..}.5WG.
```

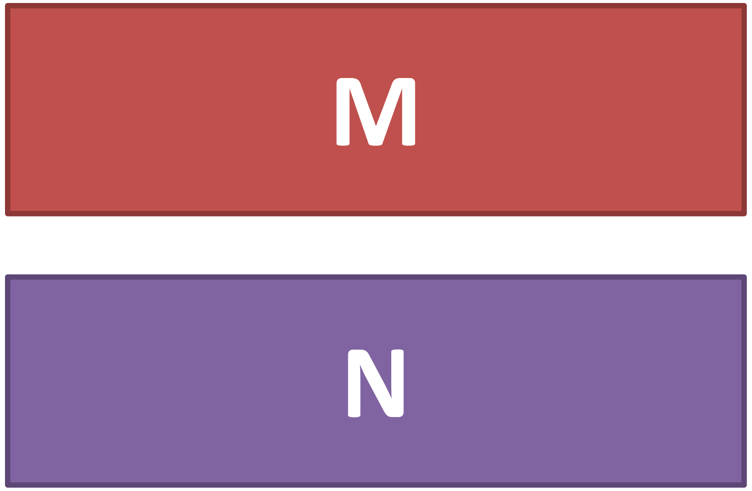
Our prefix is exactly 64 bytes

→ No padding is added!

Hash Collisions (md5collgen)



Hash Collisions (Suffix Extension)

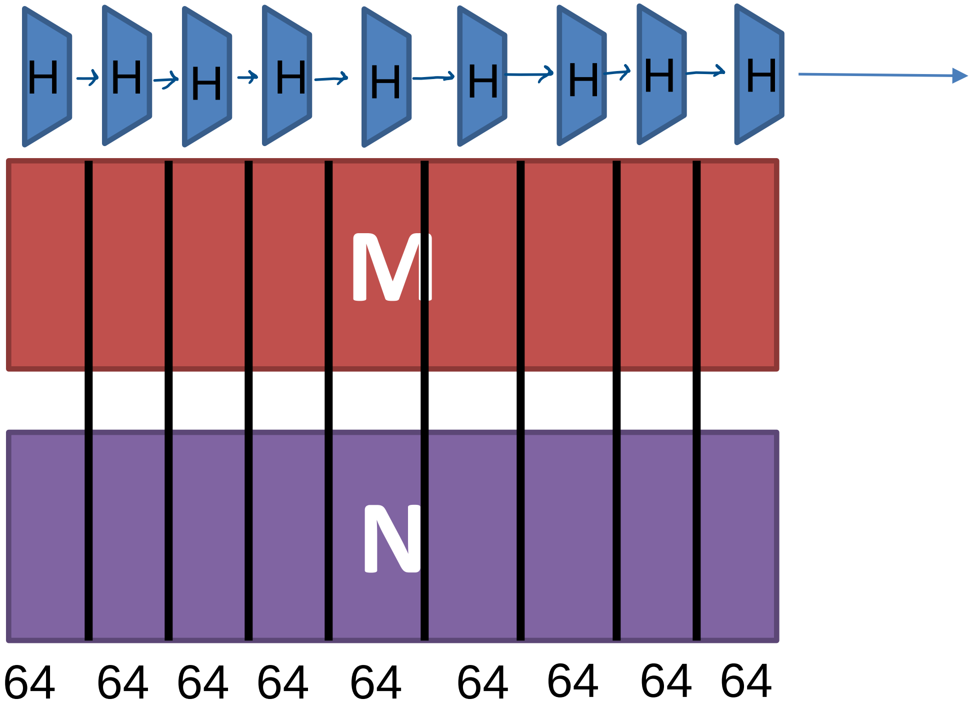


M

N

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

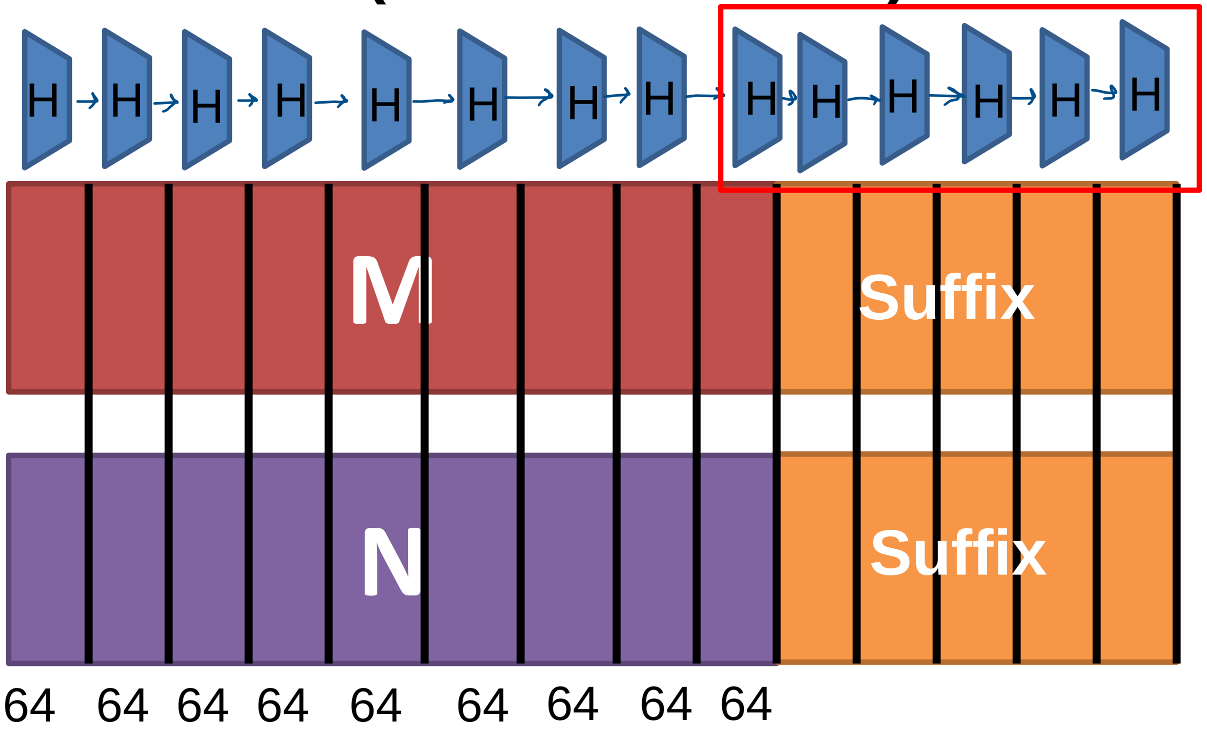
Hash Collisions (Suffix Extension)



Final Hash
(same for each)

H(m) == H(n)

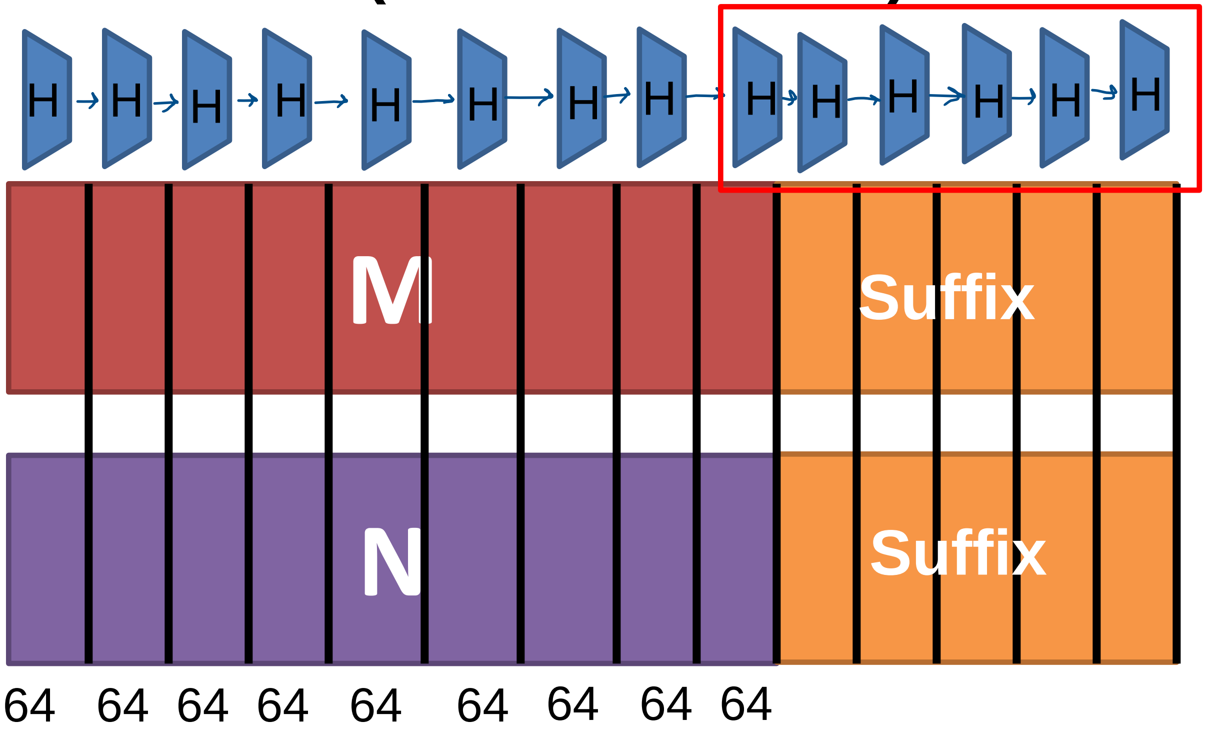
Hash Collisions (Suffix Extension)



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

H(m) == H(n)

Hash Collisions (Suffix Extension)



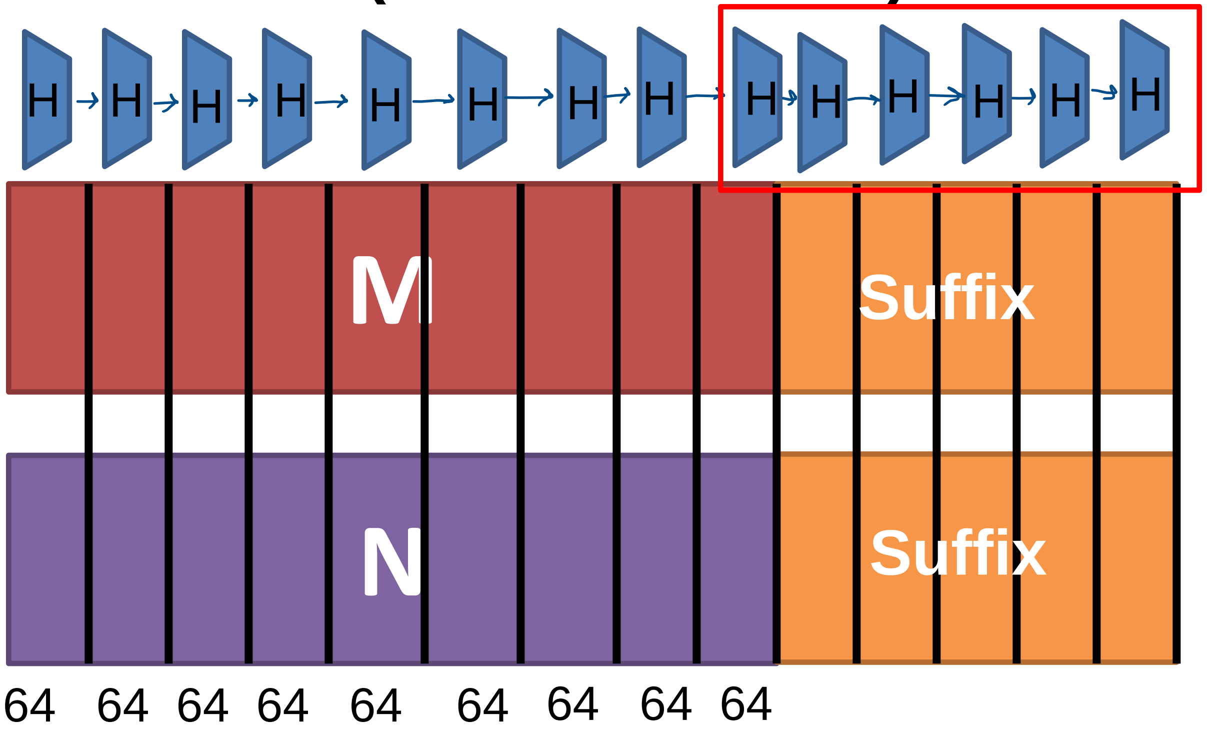
If we append the same suffix, then this computation will also be the exact same for M and N

H(m) == H(n)

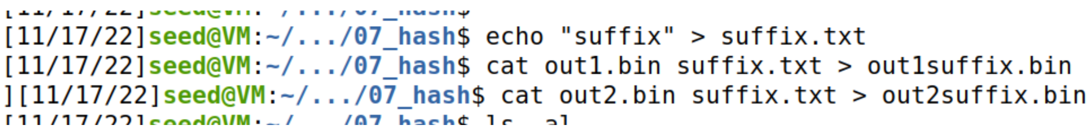
H(m || s) == H(n || s)

s = 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

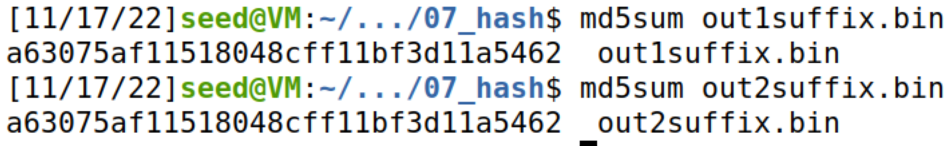


```
[11/17/22]seed@VM:~/.../07_hash$ echo "suffix" > suffix.txt
[11/17/22]seed@VM:~/.../07_hash$ cat out1.bin suffix.txt > out1suffix.bin
][11/17/22]seed@VM:~/.../07_hash$ cat out2.bin suffix.txt > out2suffix.bin
```

H(m) == H(n)

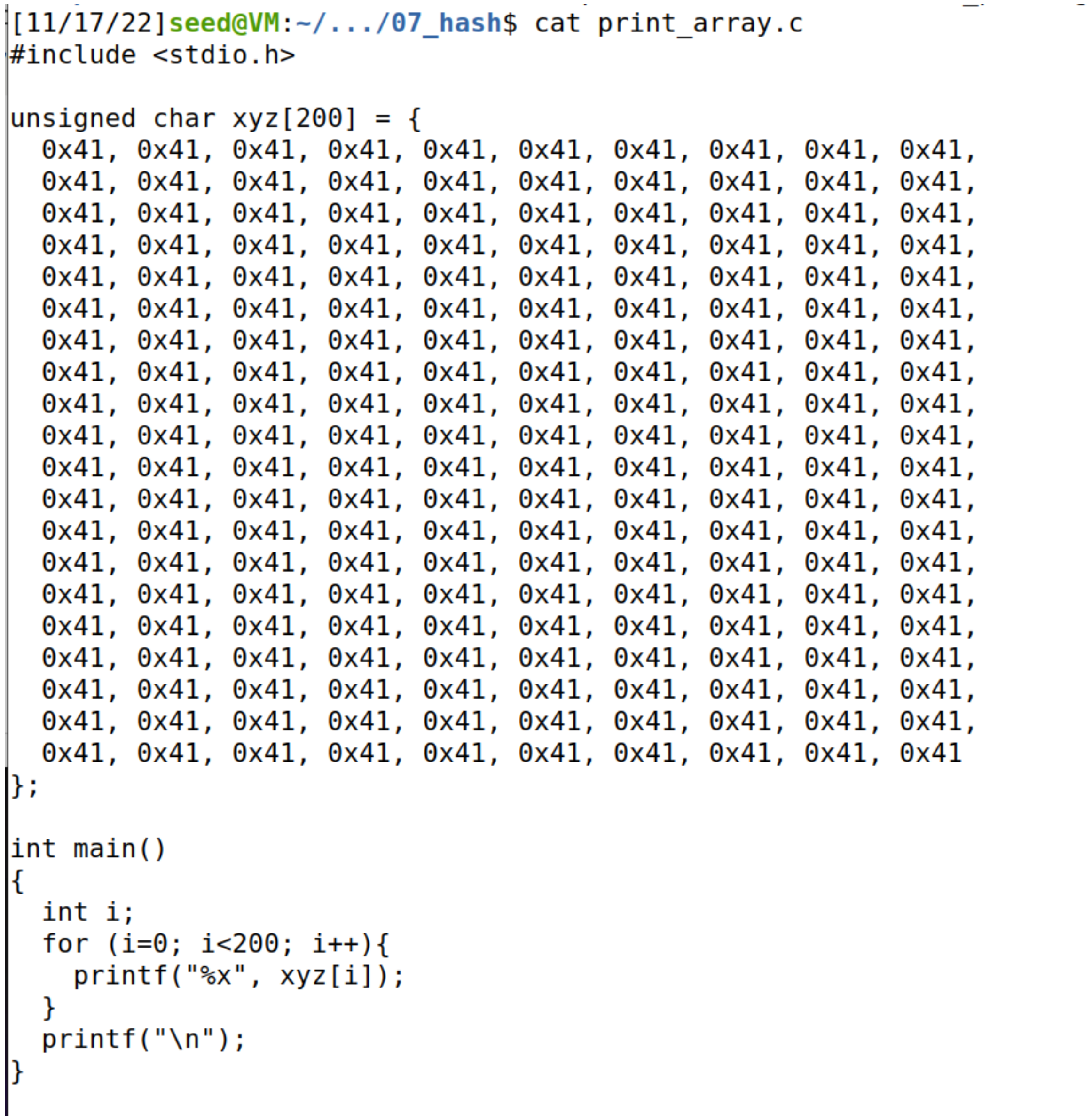
H(m || s) == H(n || s)

s = shared suffix



```
[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
```

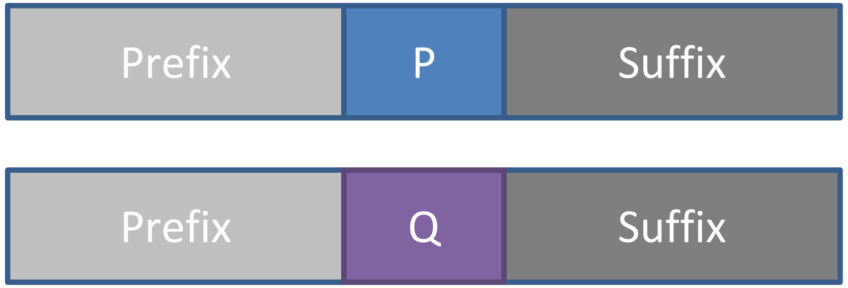
Hash Collisions (Generating Two executable files with the same MD5 hash)



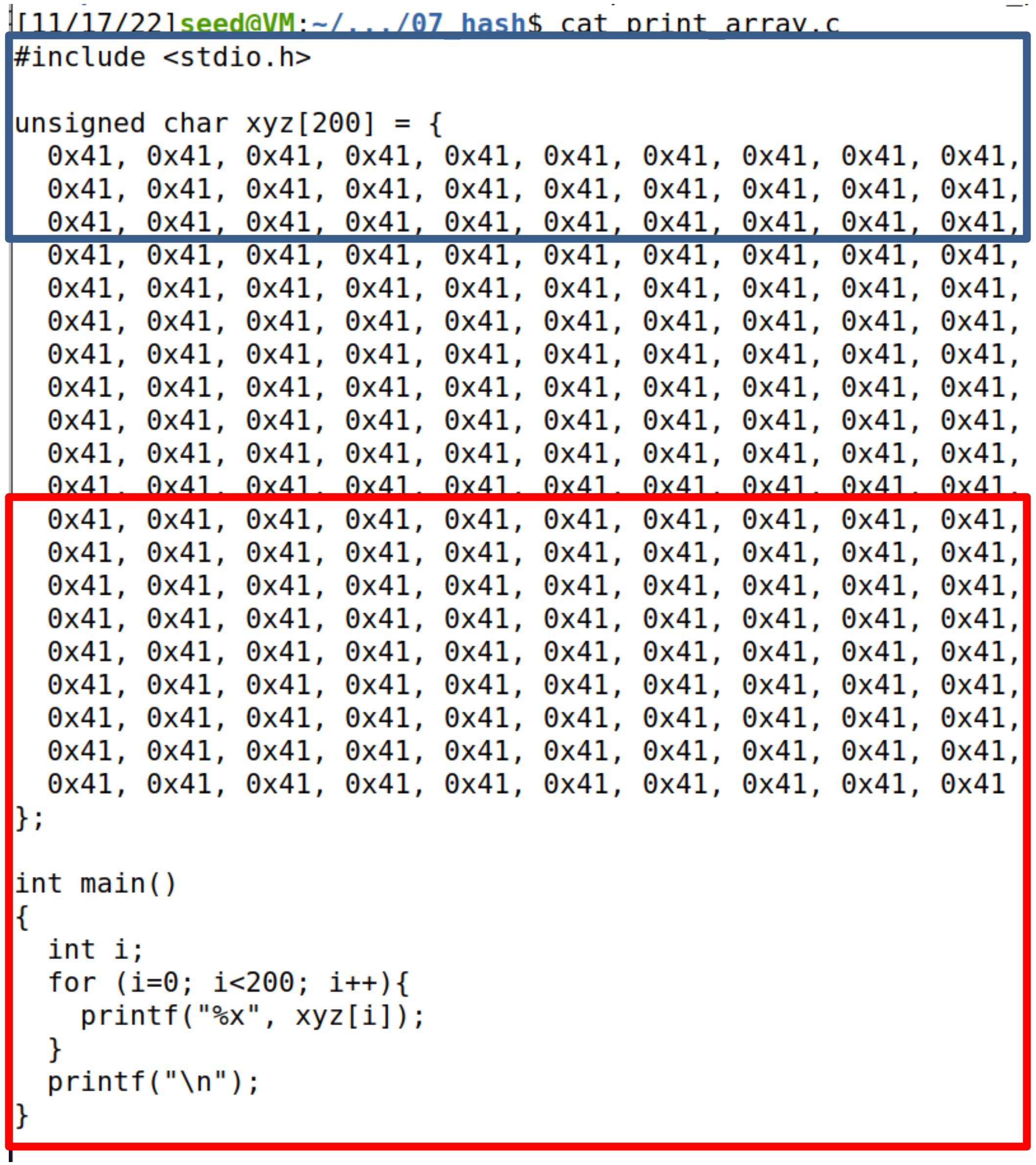
```
[11/17/22]seed@VM:~/.../07_hash$ cat print_array.c
#include <stdio.h>

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

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



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



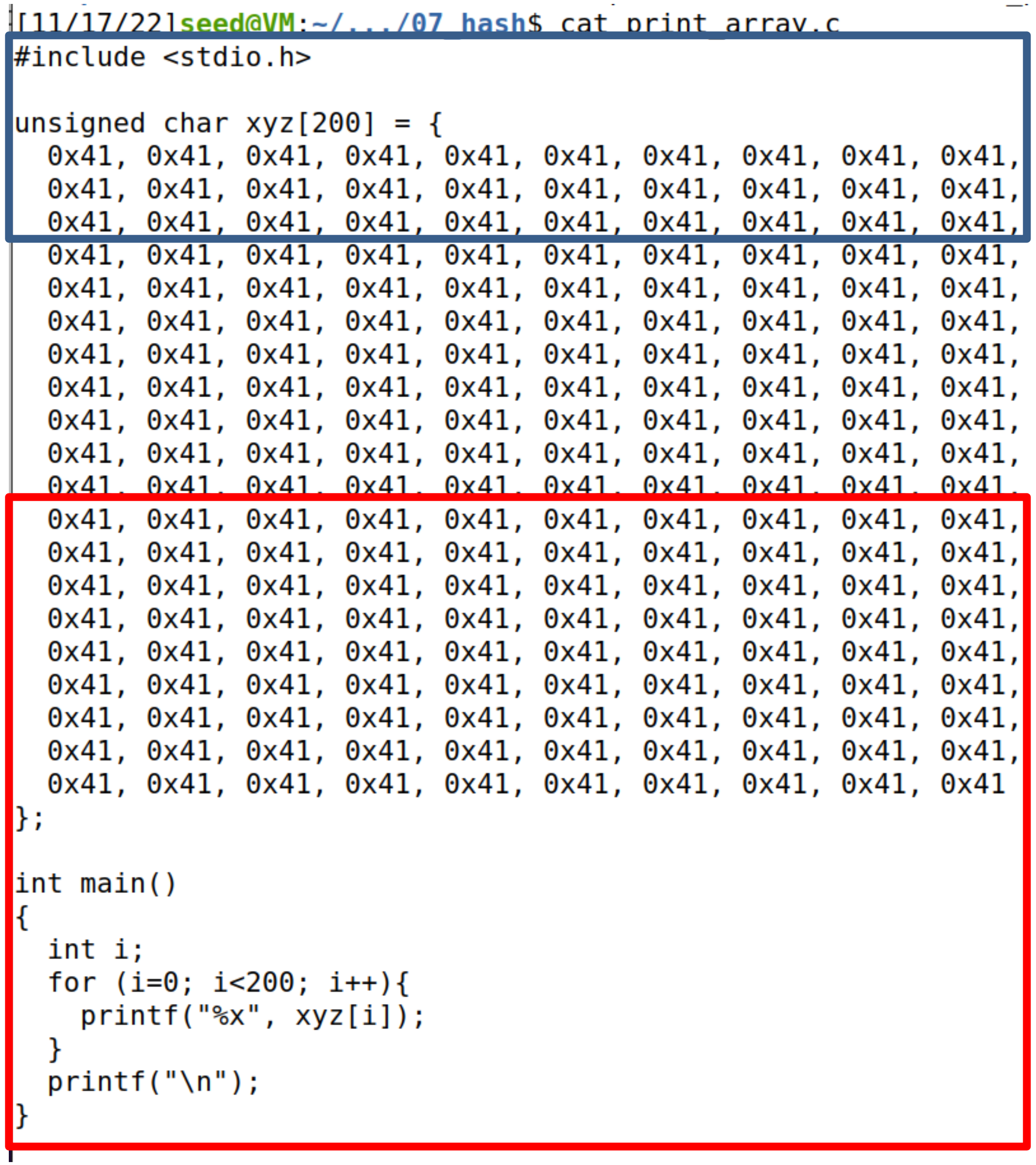
```
[11/17/22]seed@VM:~/.../07_hash$ cat print_array.c
#include <stdio.h>

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

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

Prefix

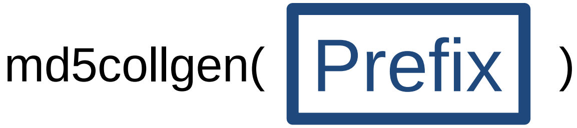
Suffix

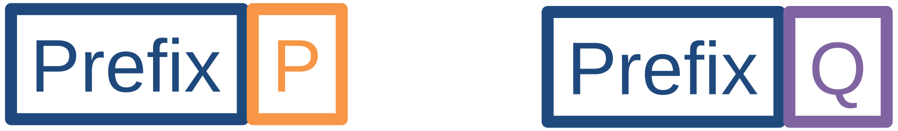
Hash Collisions (Generating Two executable files with the same MD5 hash)



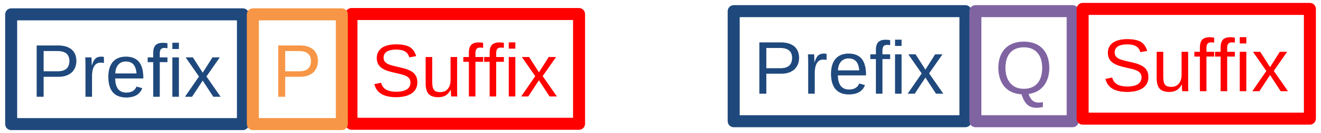
```
[11/17/22]seed@VM:~/.../07_hash$ cat print_array.c
#include <stdio.h>

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,
  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");
}
```

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



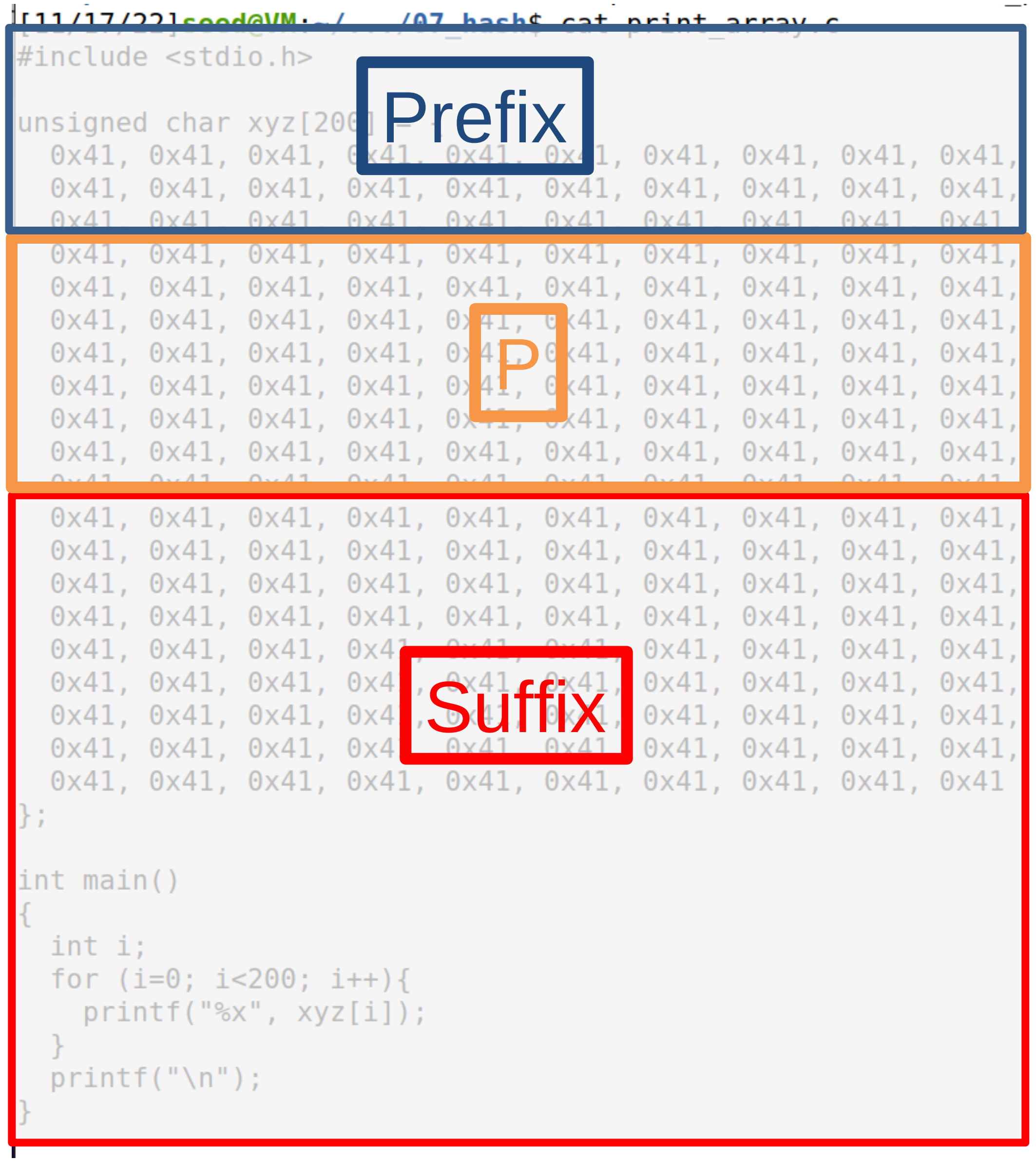


These will have the same hash!
P and Q will be 128 bytes (multiple of 64)

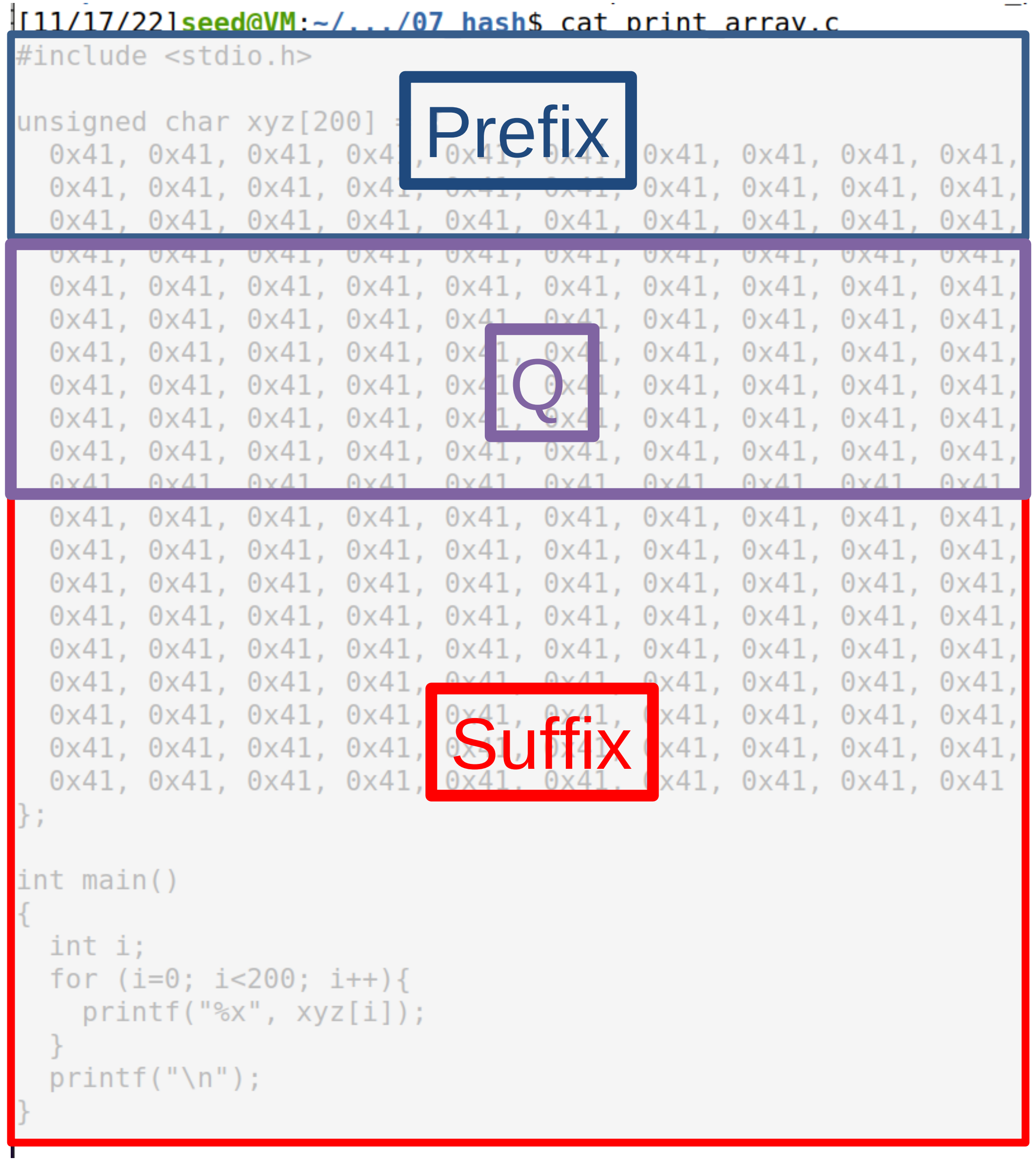


Because we know the suffix extension property holds true, we know the hash of these two programs will also be the same

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

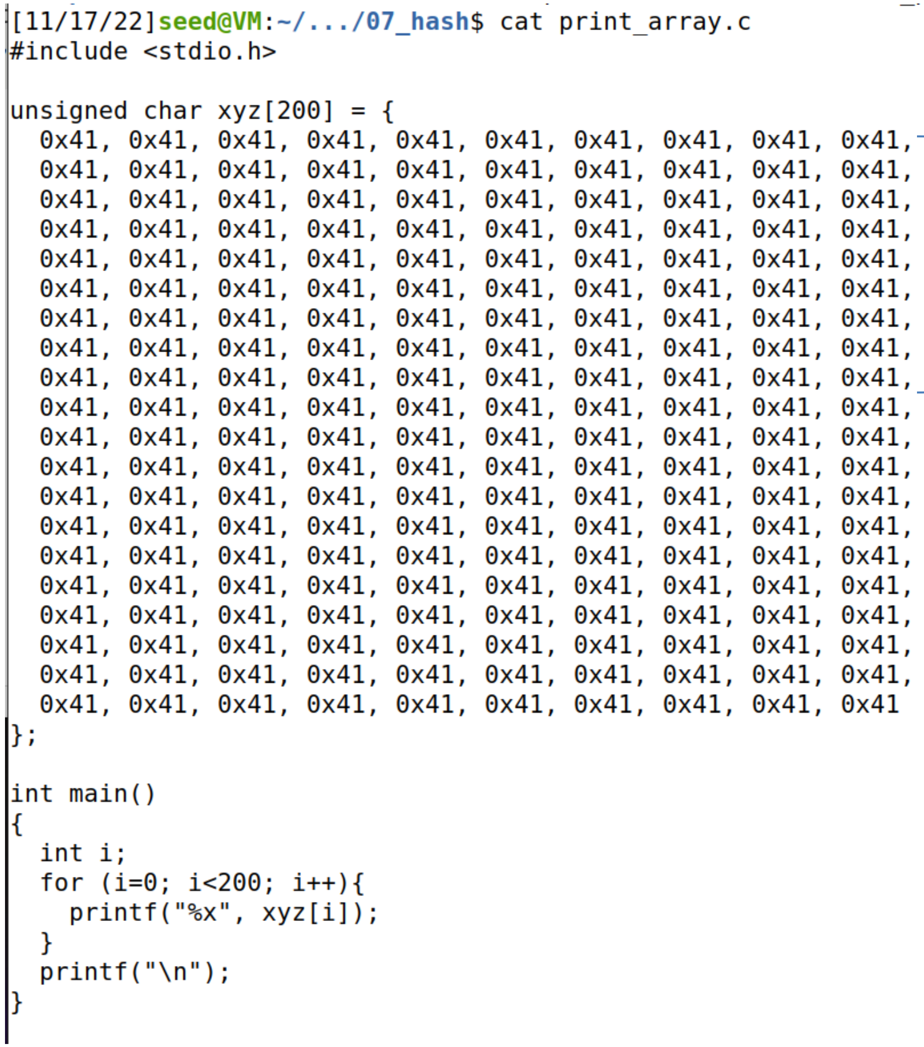


```
[11/17/22]seed@VM:~/.../07_hash$ cat print_array.c
#include <stdio.h>

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



```
[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,
  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");
}
```

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

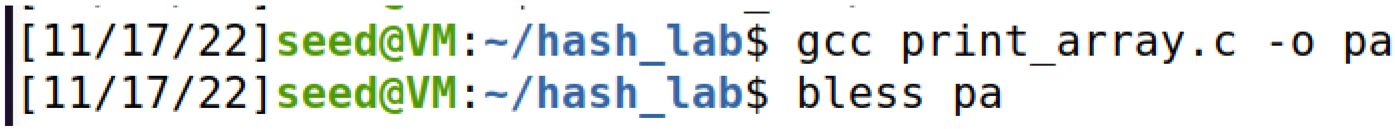


```
[11/17/22]seed@VM:~/.../07_hash$ cat print_array.c
#include <stdio.h>

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

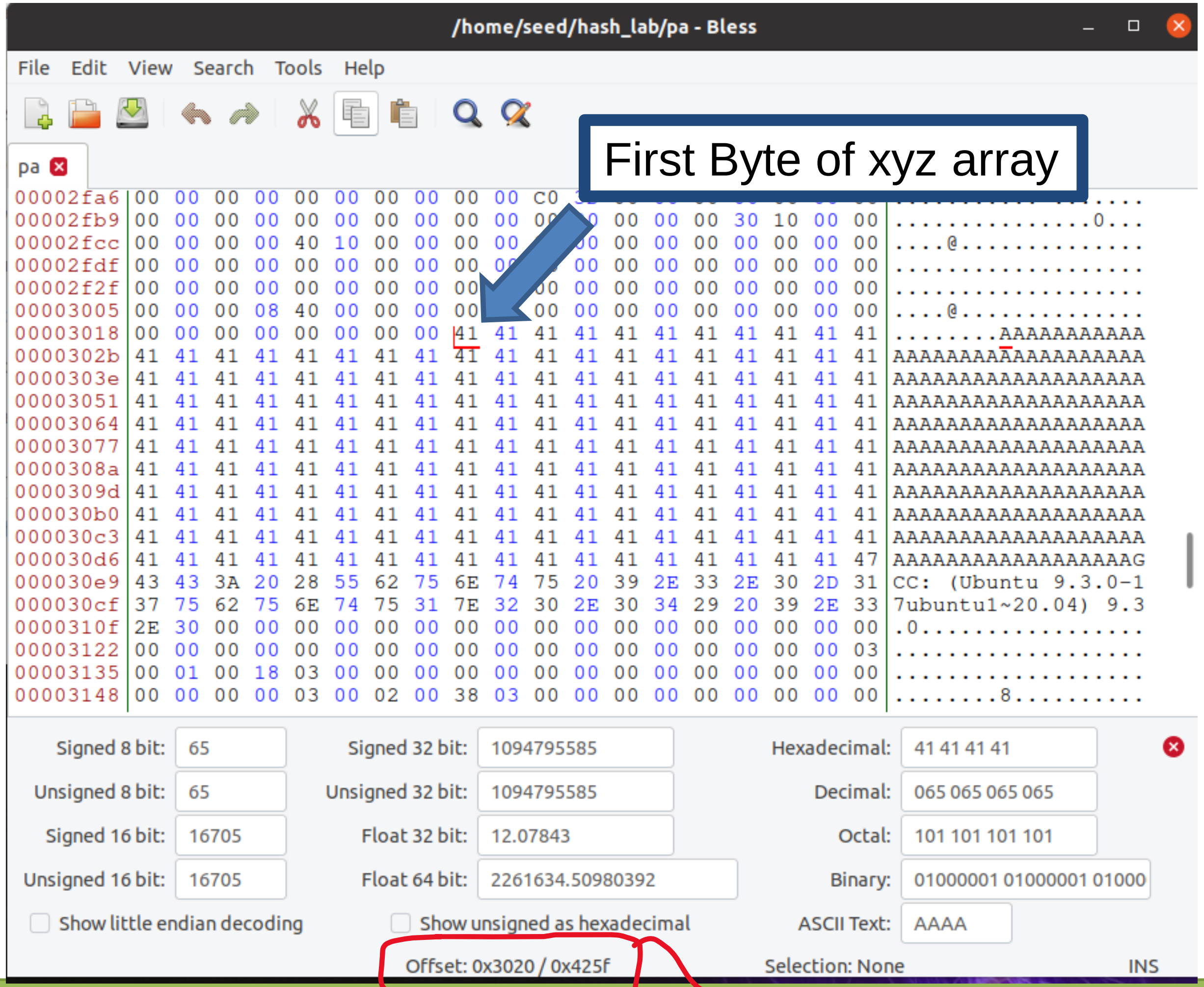
We can change the contents of this section of the program because it is just array data (it won’t break anything)

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
```



/home/seed/hash_lab/pa - Bless
File Edit View Search Tools Help
pa

First Byte of xyz array

```
00002fa6 00 00 00 00 00 00 00 00 00 00 C0
00002fb9 00 00 00 00 00 00 00 00 00 00 00       00 00 00 30 10 00 00 .................0...
00002fcc 00 00 00 00 40 10 00 00 00 00          00 00 00 00 00 00 00 00 ....@..............
00002fdf 00 00 00 00 00 00 00 00 00 00          00 00 00 00 00 00 00 00 ...................
00002f2f 00 00 00 00 00 00 00 00 00          00 00 00 00 00 00 00 00 00 ...................
00003005 00 00 00 08 40 00 00 00 00          00 00 00 00 00 00 00 00 00 ....@..............
00003018 00 00 00 00 00 00 00 00 41 41 41 41 41 41 41 41 41 41 41 ........AAAAAAAAAAA
0000302b 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAAA
0000303e 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAAA
00003051 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAAA
00003064 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAAA
00003077 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAAA
0000308a 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAAA
0000309d 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAAA
000030b0 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAAA
000030c3 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAAA
000030d6 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 47 AAAAAAAAAAAAAAAAAAG
000030e9 43 43 3A 20 28 55 62 75 6E 74 75 20 39 2E 33 2E 30 2D 31 CC: (Ubuntu 9.3.0-1
000030cf 37 75 62 75 6E 74 75 31 7E 32 30 2E 30 34 29 20 39 2E 33 7ubuntu1~20.04) 9.3
0000310f 2E 30 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .0.................
00003122 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 03 ...................
00003135 00 01 00 18 03 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ...................
00003148 00 00 00 00 03 00 02 00 38 03 00 00 00 00 00 00 00 00 00 ........8..........
```

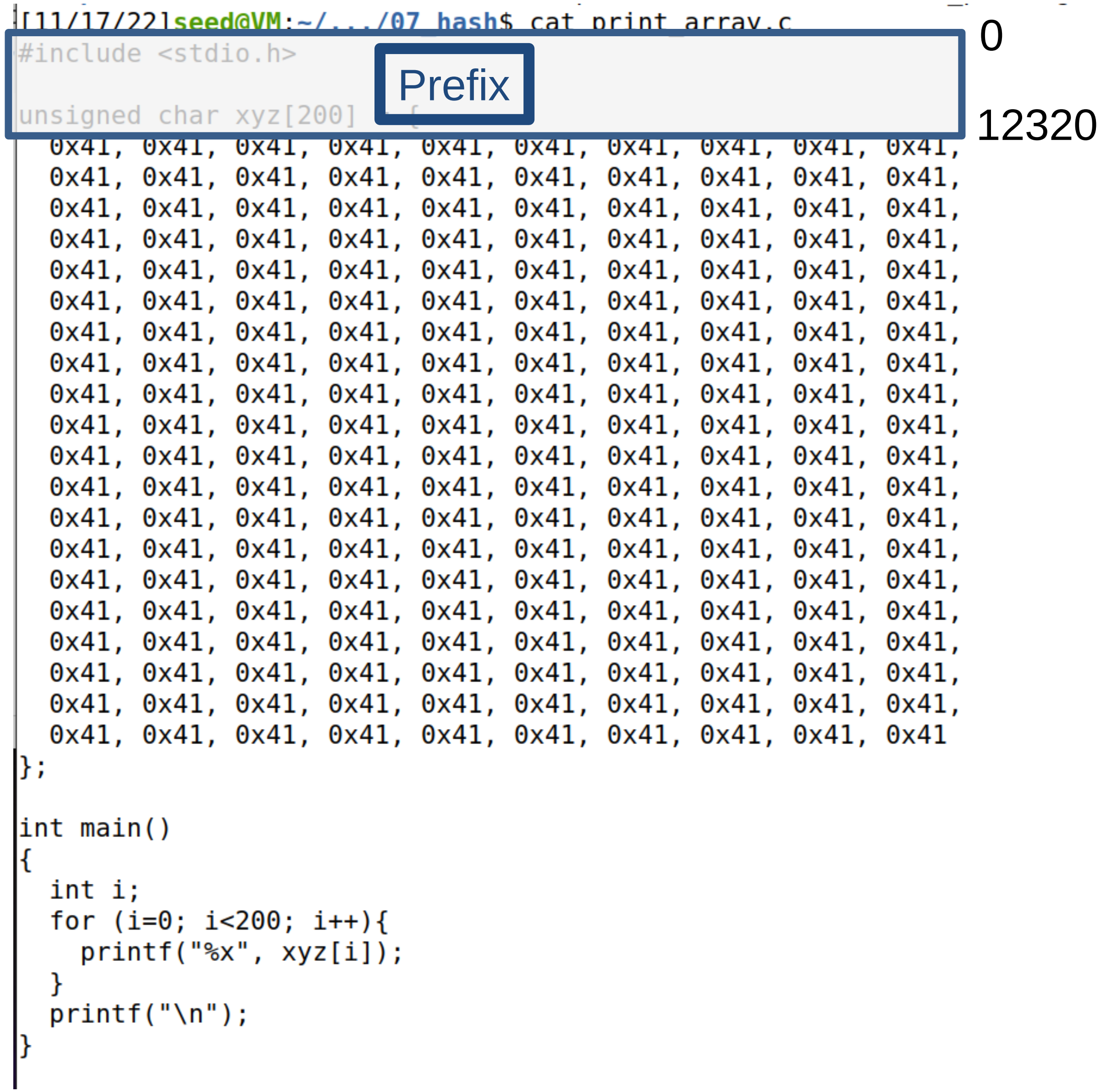
Signed 8 bit:	65	Signed 32 bit:	1094795585	Hexadecimal:	41 41 41 41
Unsigned 8 bit:	65	Unsigned 32 bit:	1094795585	Decimal:	065 065 065 065
Signed 16 bit:	16705	Float 32 bit:	12.07843	Octal:	101 101 101 101
Unsigned 16 bit:	16705	Float 64 bit:	2261634.50980392	Binary:	01000001 01000001 01000
☐ Show little endian decoding		☐ Show unsigned as hexadecimal		ASCII Text:	AAAA

Offset: 0x3020 / 0x425f   Selection: None   INS

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,
  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");
}
```

Prefix

0

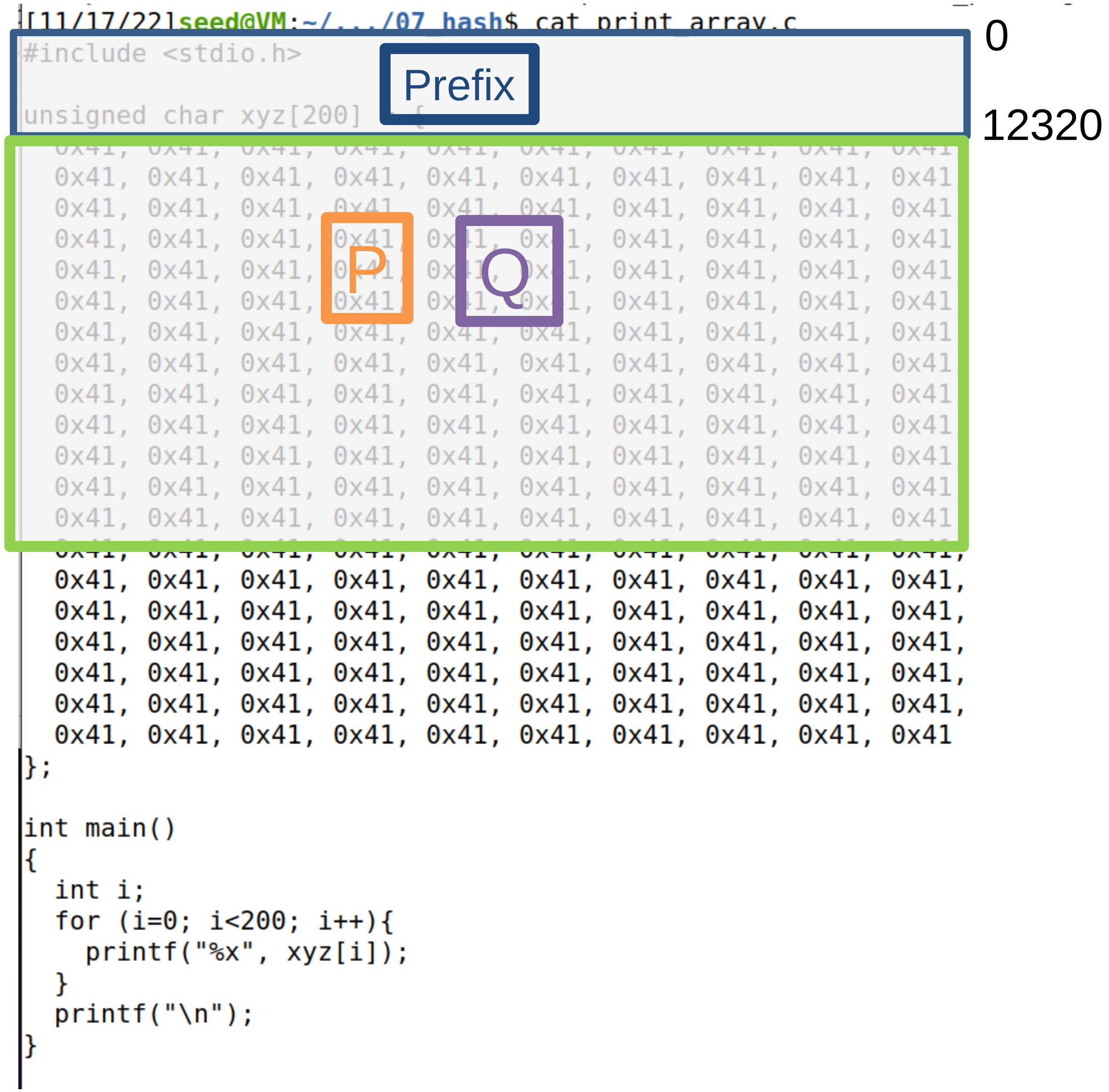
12320

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



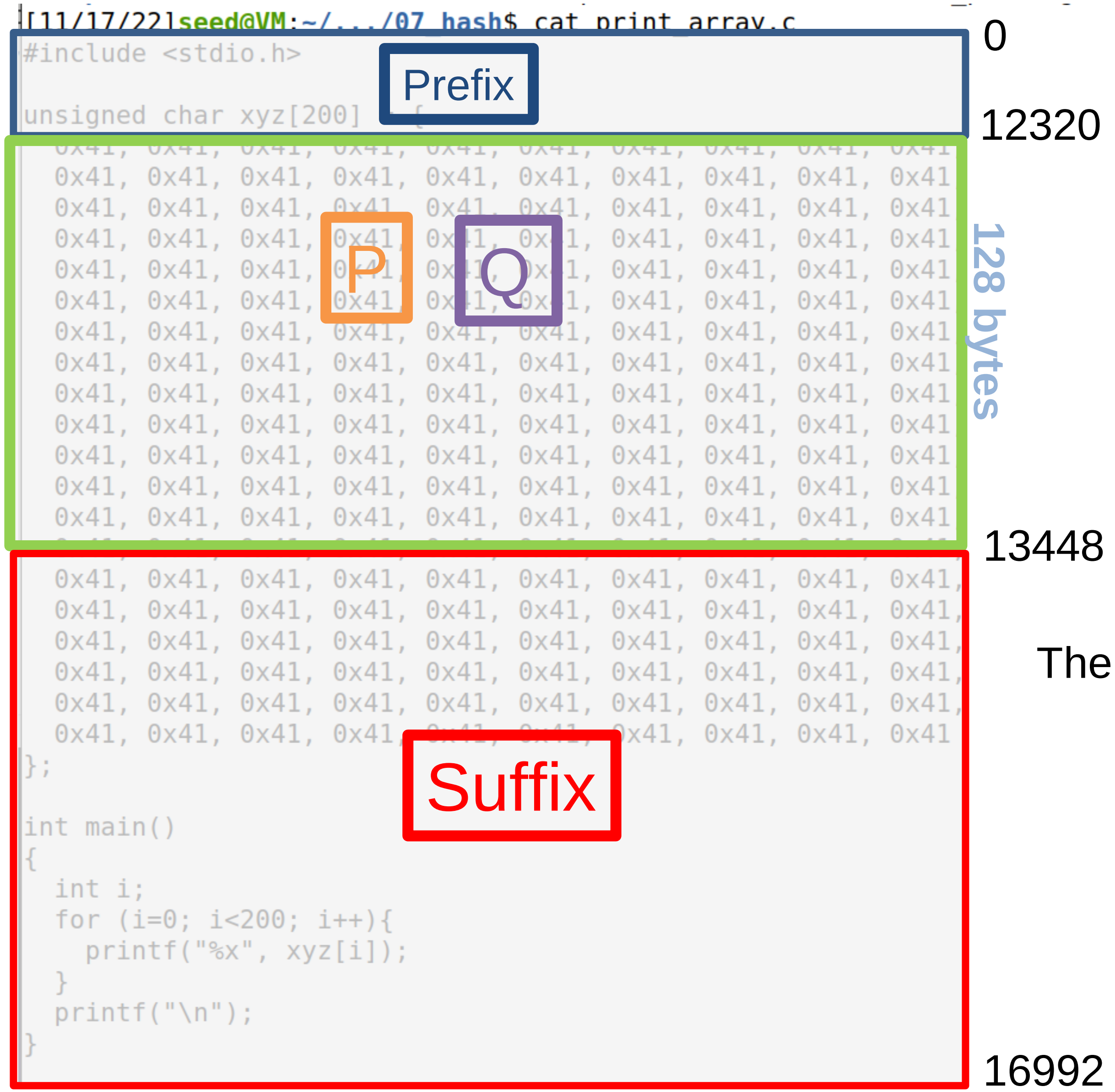
```
[11/17/22]seed@VM:~/.../07_hash$ cat print_array.c
#include <stdio.h>

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,
  0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
  0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
  0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
  0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
  0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 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



```
[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");
}
```

16992 (size of executable)

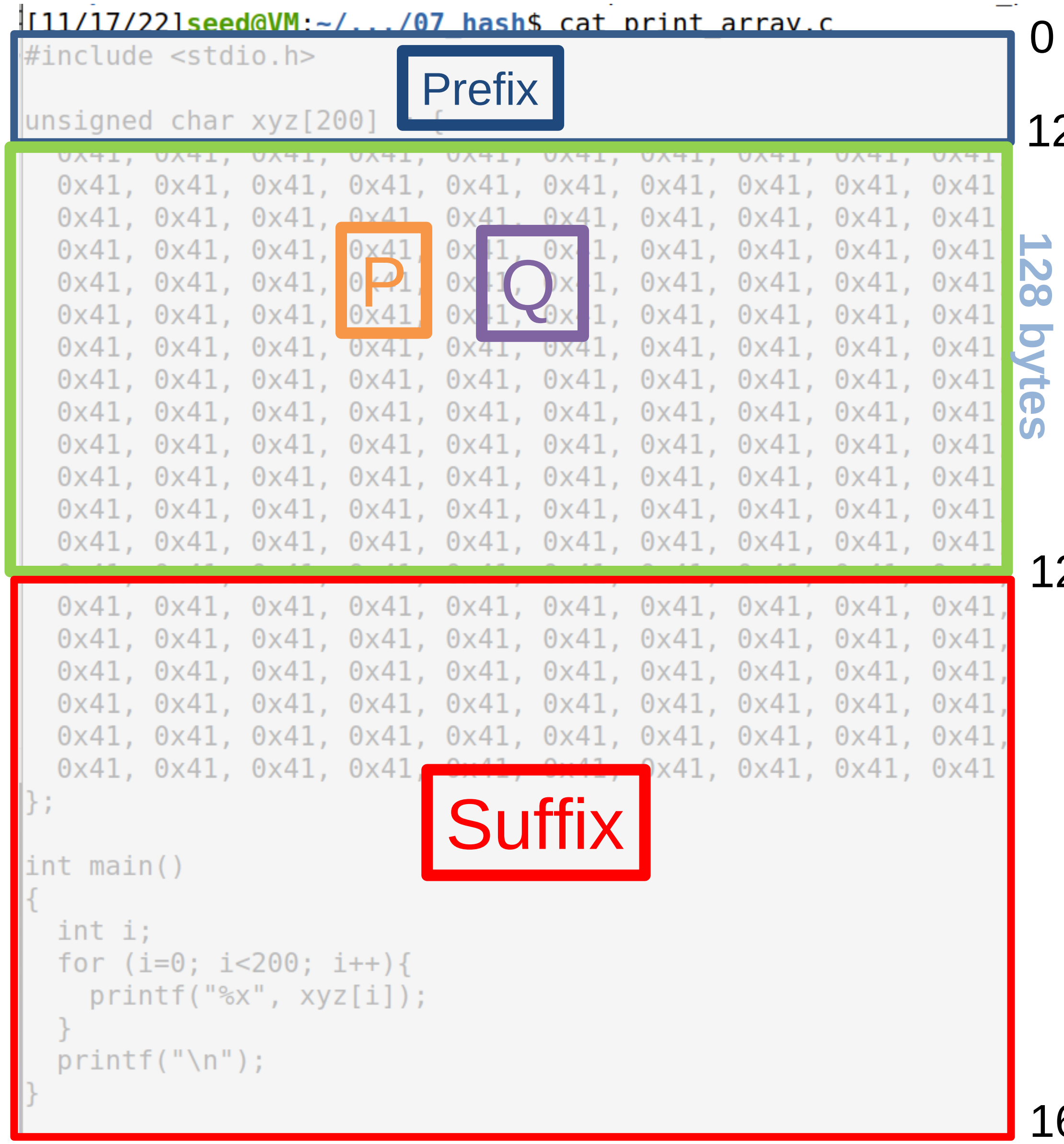
Our prefix will be bytes 0-12320 of the program!

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] = {
  0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
  0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
  0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
  0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
  0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
  0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
  0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
  0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
  0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
  0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
  0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
  0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
  0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
  0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
  0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
  0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
  0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
  0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 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");
}
```

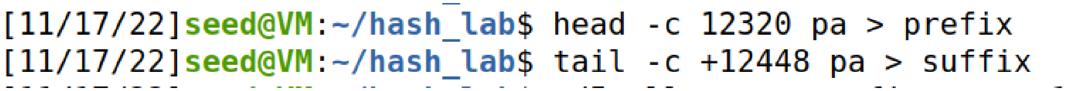
0

12320

12448

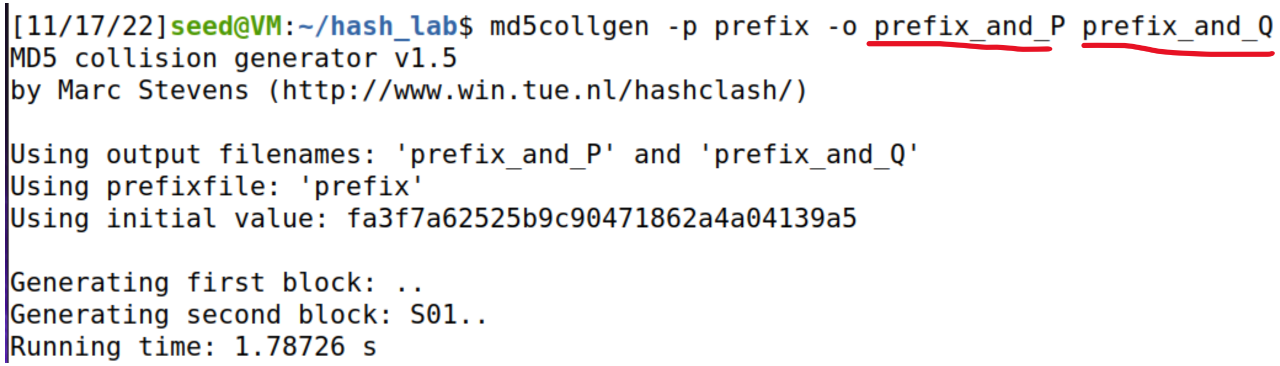
16992 (size of executable)

Get contents of prefix and suffix

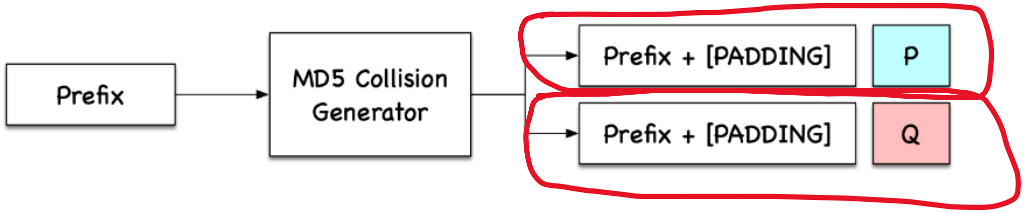


```
[11/17/22]seed@VM:~/hash_lab$ head -c 12320 pa > prefix
[11/17/22]seed@VM:~/hash_lab$ tail -c +12448 pa > suffix
```

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

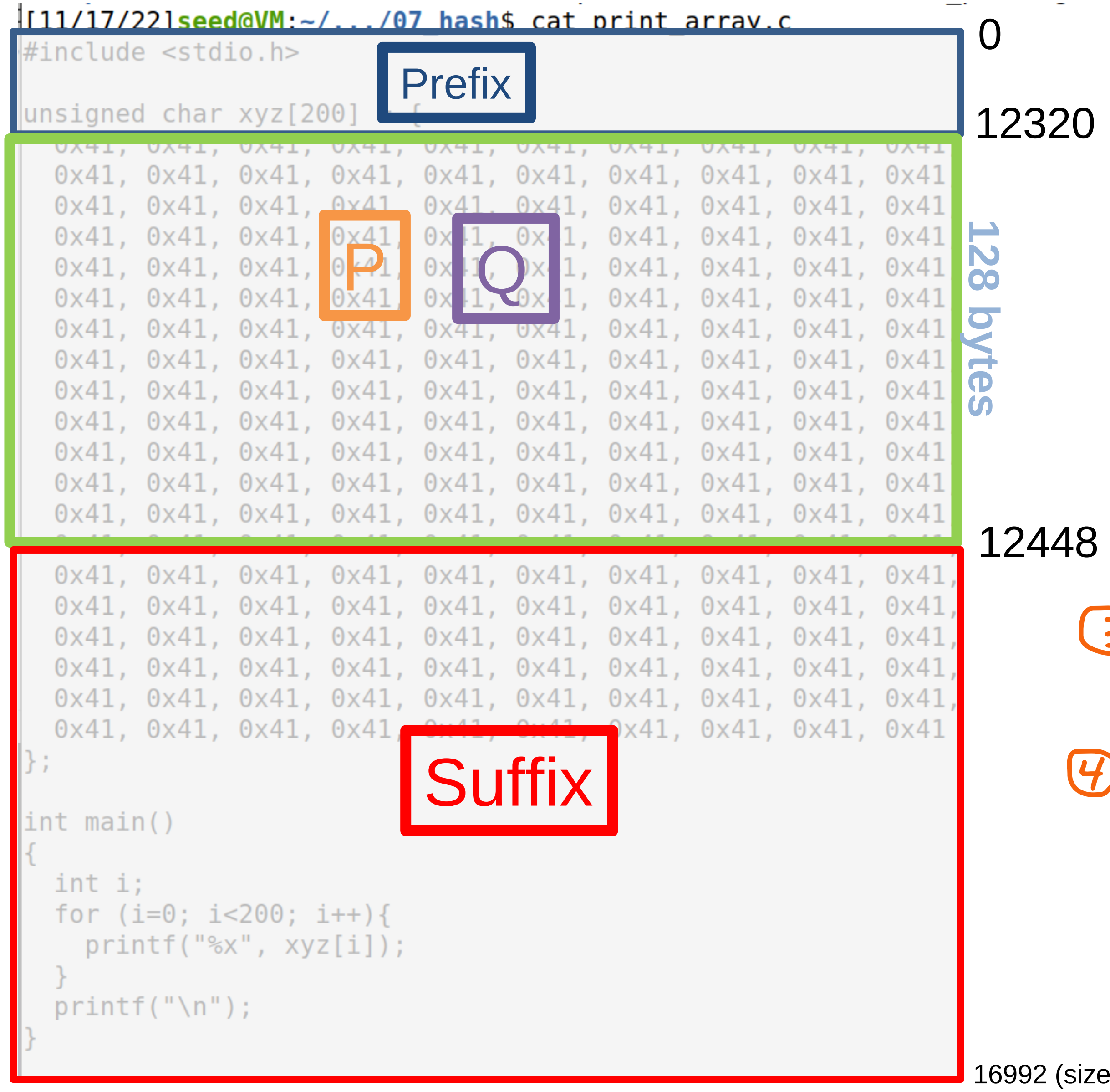


```
[11/17/22]seed@VM:~/hash_lab$ md5collgen -p prefix -o prefix_and_P prefix_and_Q
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

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

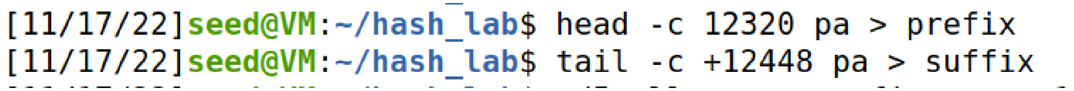


```
[11/17/22]seed@VM:~/.../07_hash$ cat print_array.c
#include <stdio.h>

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,
  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");
}
```

16992 (size of executable)

1

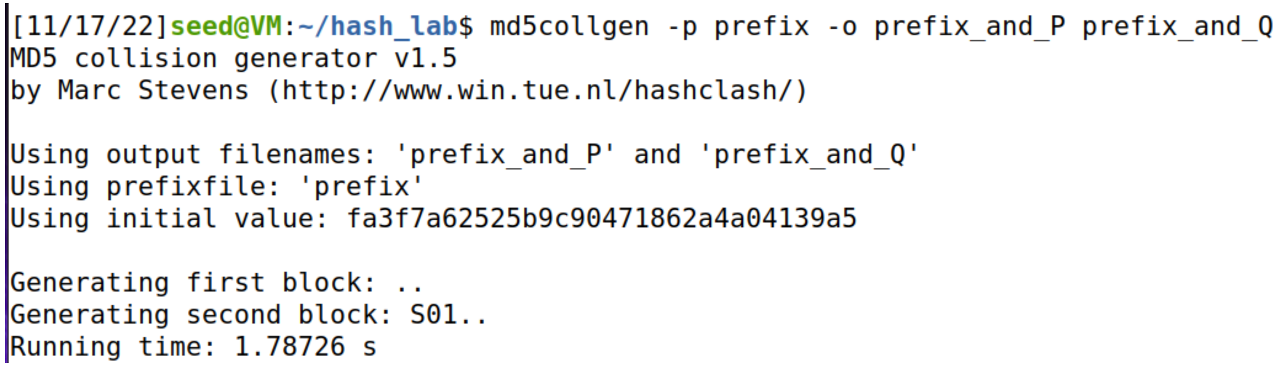
Get contents of prefix and suffix



```
[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

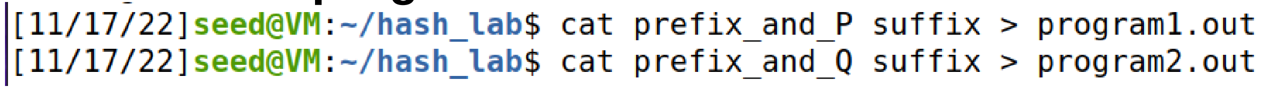
Use collision tool to get (prefix + P) and (prefix + Q)



```
[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

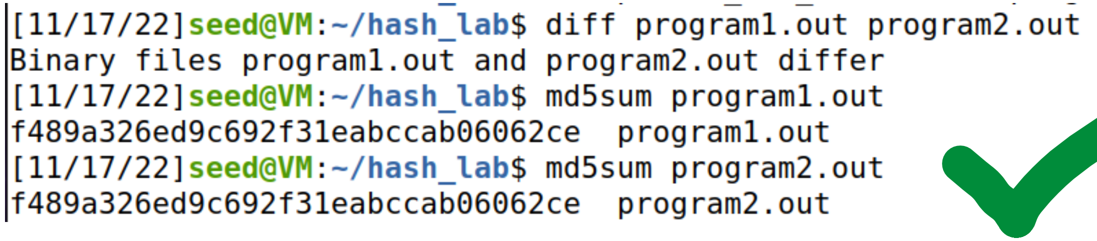
Add suffix to programs



```
[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
```

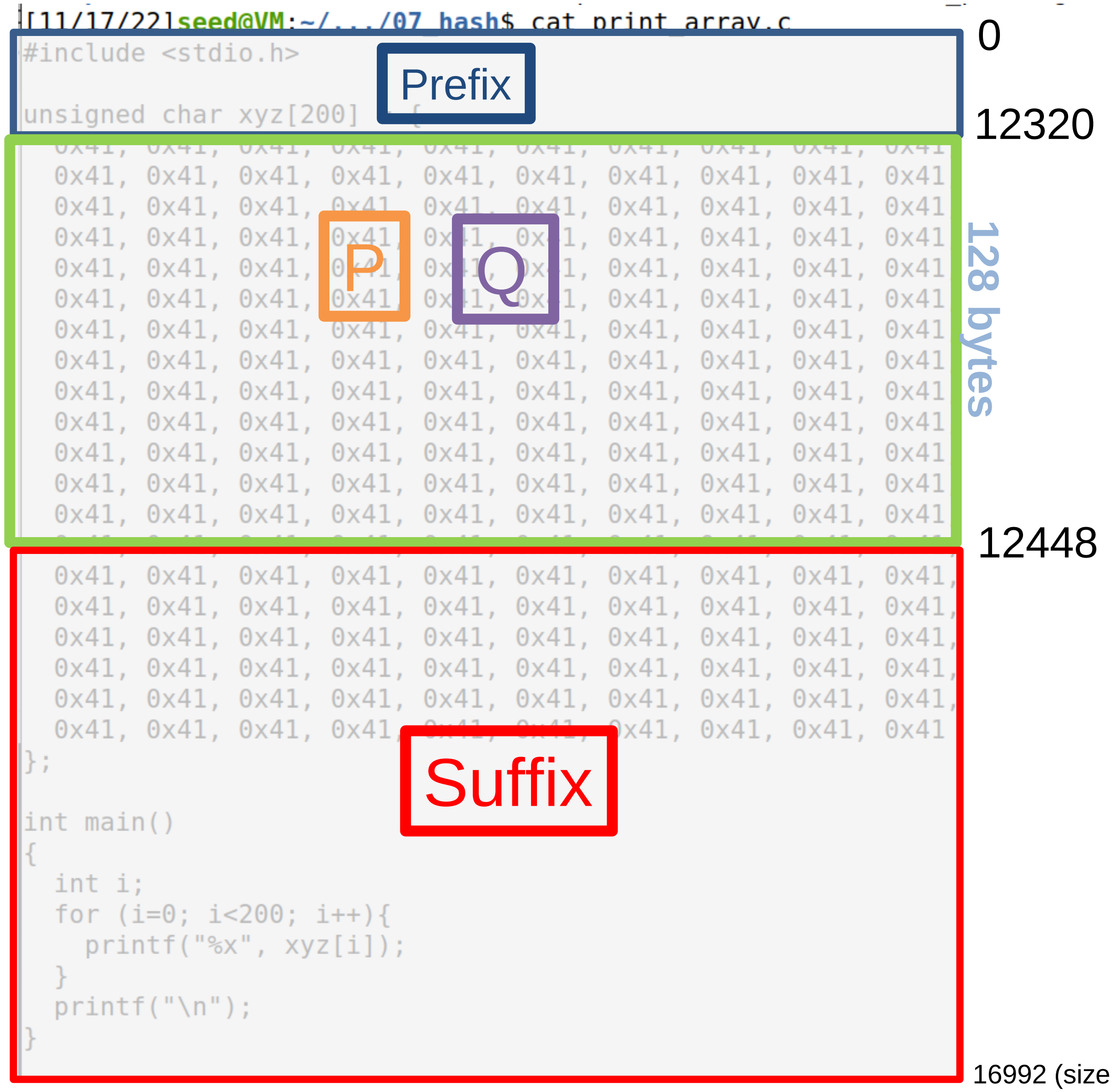
4

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
```

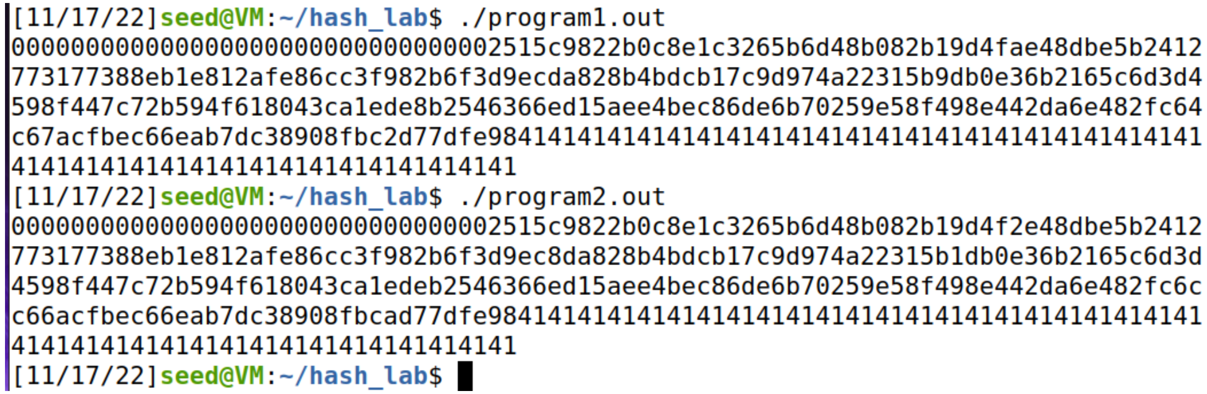
Task 4 on the lab



```
[11/17/22]seed@VM:~/.../07_hash$ cat print_array.c
#include <stdio.h>

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,
  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");
}
```

5

Make sure you still have a valid program ☺



```
[11/17/22]seed@VM:~/hash_lab$ ./program1.out
00000000000000000000000000000002515c9822b0c8e1c3265b6d48b082b19d4fae48dbe5b2412
773177388eb1e812afe86cc3f982b6f3d9ecda828b4bdcb17c9d974a22315b9db0e36b2165c6d3d4
598f447c72b594f618043ca1ede8b2546366ed15aee4bec86de6b70259e58f498e442da6e482fc64
c67acfbec66eab7dc38908fbc2d77dfe984141414141414141414141414141414141414141414141
41414141414141414141414141414141
[11/17/22]seed@VM:~/hash_lab$ ./program2.out
00000000000000000000000000000002515c9822b0c8e1c3265b6d48b082b19d4f2e48dbe5b2412
773177388eb1e812afe86cc3f982b6f3d9ec8da828b4bdcb17c9d974a22315b1db0e36b2165c6d3d
4598f447c72b594f618043ca1edeb2546366ed15aee4bec86de6b70259e58f498e442da6e482fc6c
c66acfbec66eab7dc38908fbcad77dfe984141414141414141414141414141414141414141414141
41414141414141414141414141414141
[11/17/22]seed@VM:~/hash_lab$ 
```

Somewhere in this output, you should find a small difference