CSCI 476: Computer Security

Hashing (Part 1)

Reese Pearsall

Fall 2024

Announcements

Project due 11/21

- Message me about your project idea if you haven’t done so already

Lab 8 (Secret Key Encryption) Due 11/24

I’ll be back on Thursday

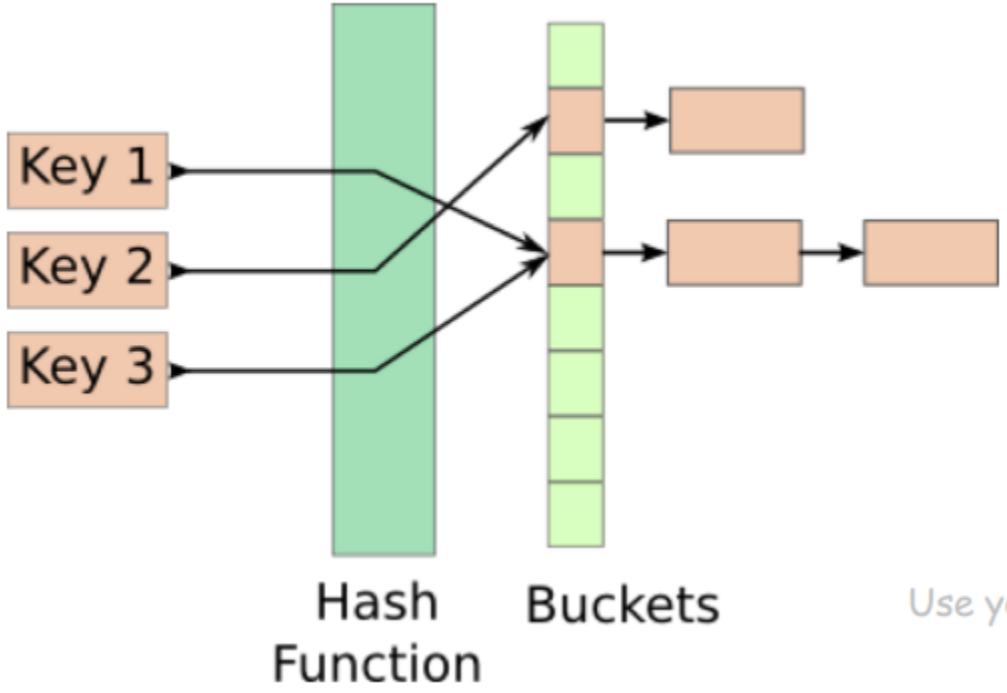
Lab 8

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**



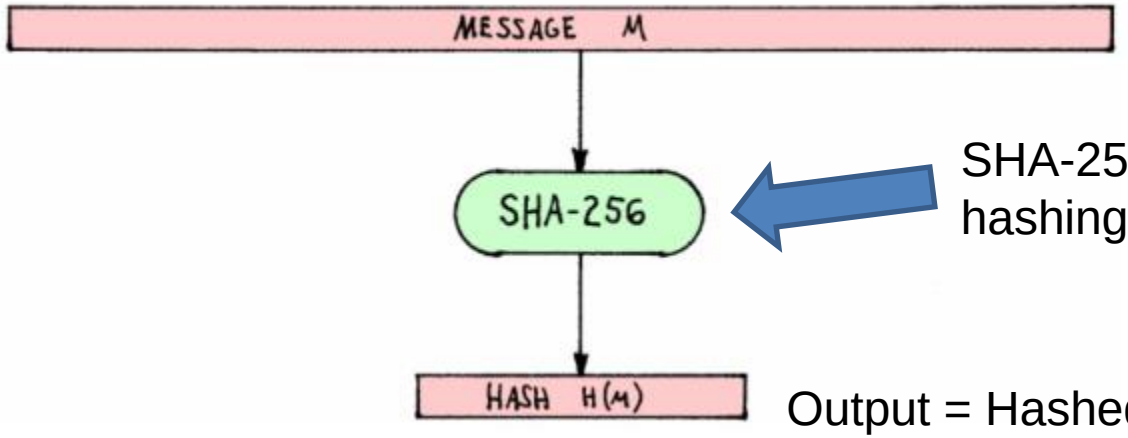
How many buckets?
What to do if two keys map to the same bucket?

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



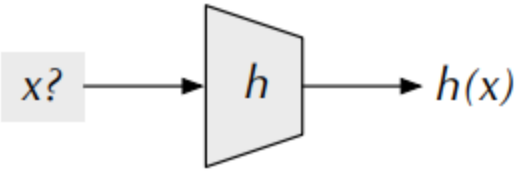
Input = Message M

SHA-256 = A cryptographically secure hashing function

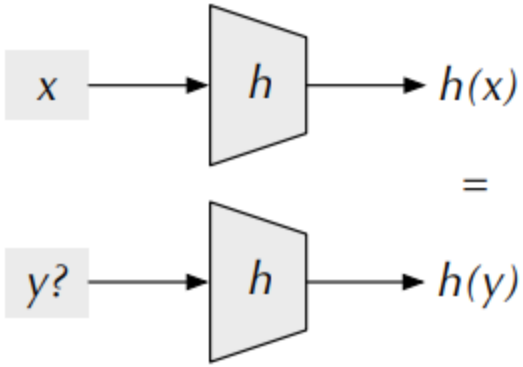
Output = Hashed Message M

Hash Functions Properties

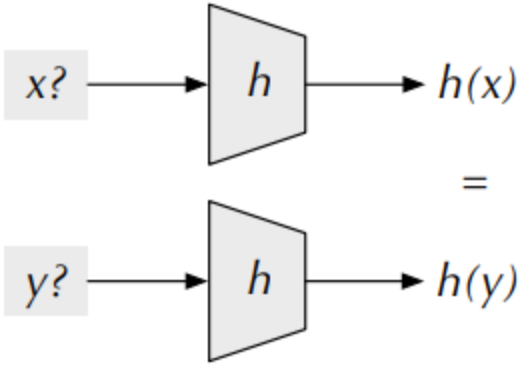
- **Preimage Resistance ("One-Way")**
  Given h(x) = z, hard to find x
  (or any input that hashes to z for that matter)



- **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

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

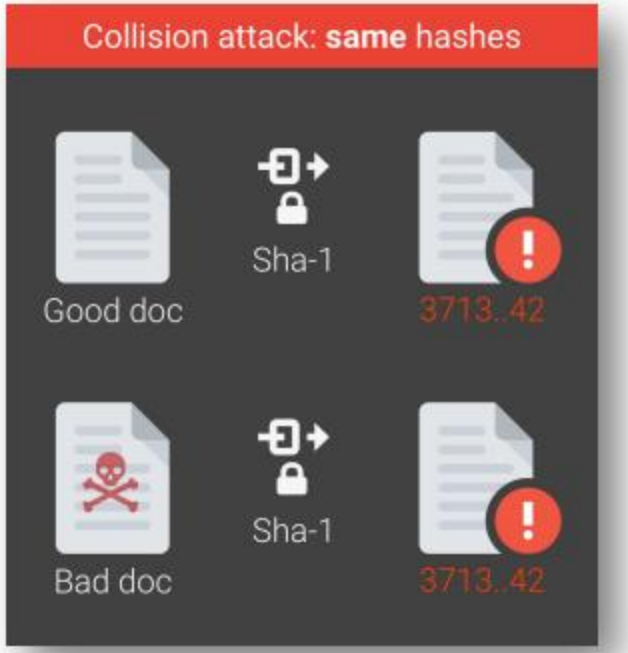
- Status of Algorithms:
  - MD2, MD4 - severely broken (obsolete)
  - MD5 - collision resistance property broken; one-way property not broken
    - Often used for file integrity checking
    - No longer recommended for use!
  - MD6 - developed in response to proposal by NIST
    - Not widely used…

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

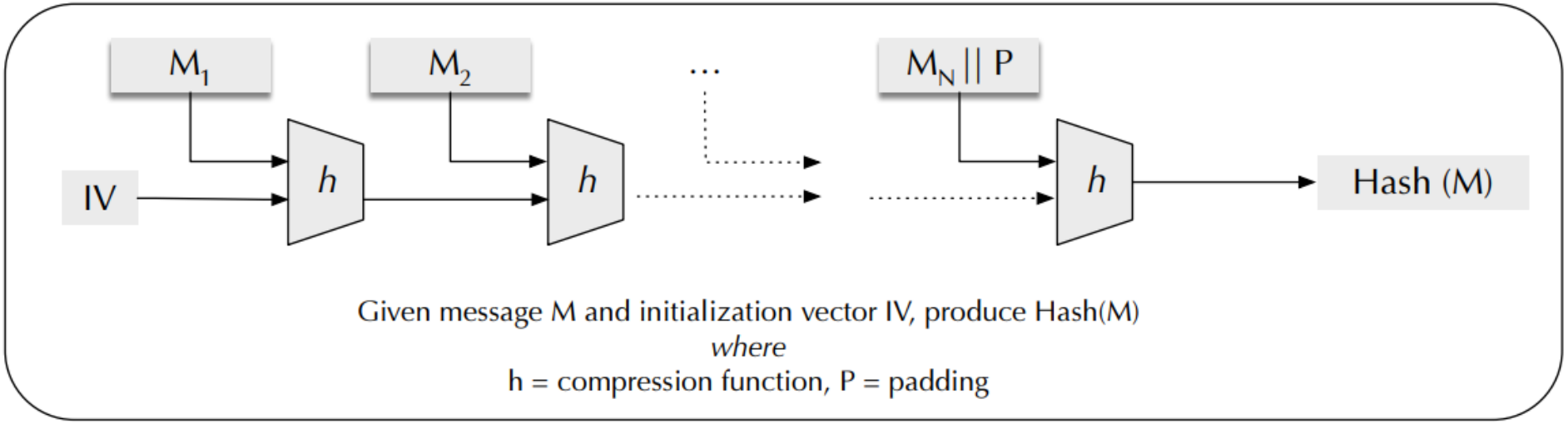
- Status of Algorithms:
  - SHA-0: withdrawn due to flaw
  - SHA-1: Designed by NSA Collision attack found in 2017
  - SHA-2: Designed by NSA
    - Includes SHA-256 and SHA-512 + other truncated versions;
    - No significant attack found yet…
  - SHA-3: Not Designed by NSA
    - Released in 2015; not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



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

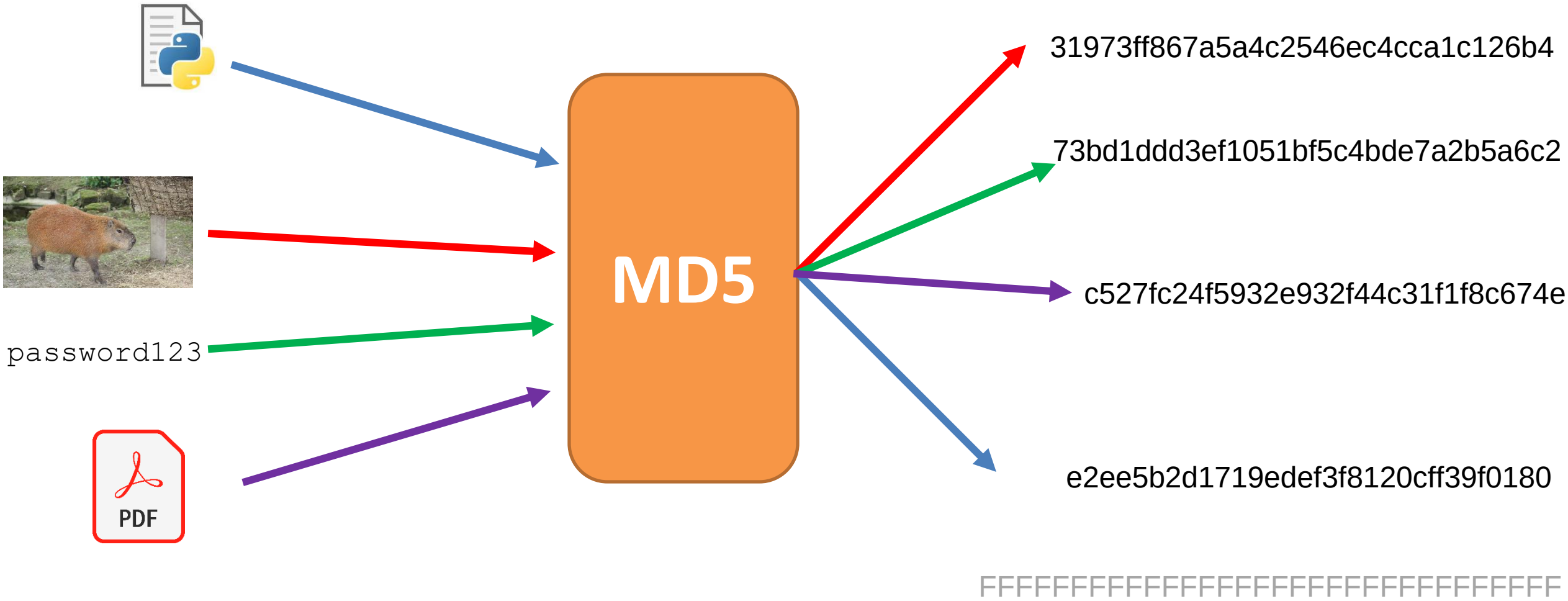
Calculating Hashes in Programming Languages

```
# 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)



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

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

```
#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

A CSCI 112 Student

hello_world

89defae676abd3e3a42b41df17c40096

```
#include <unistd.h>

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

hello_world

What if the message got tampered with?

1. Generate hash for source file

Instructor

hello_world

Applications of Hashing

Integrity Verification

```
#include <unistd.h>

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

A CSCI 112 Student

hello_world

89defae676abd3e3a42b41df17c40096

hello_world

What if the message got tampered with?

Instructor

hello_world

b0608c4e1775ad8f92e7b5c191774c5d

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

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!

Applications of Hashing

Integrity Verification

hello_world

89defae676abd3e3a42b41df17c40096

A CSCI 112 Student

hello_world

b0608c4e1775ad8f92e7b5c191774c5d

Instructor

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: f3d2227c922192656794000 64a0a1287
- VM Manual: follow this manual to install the VM on your computer

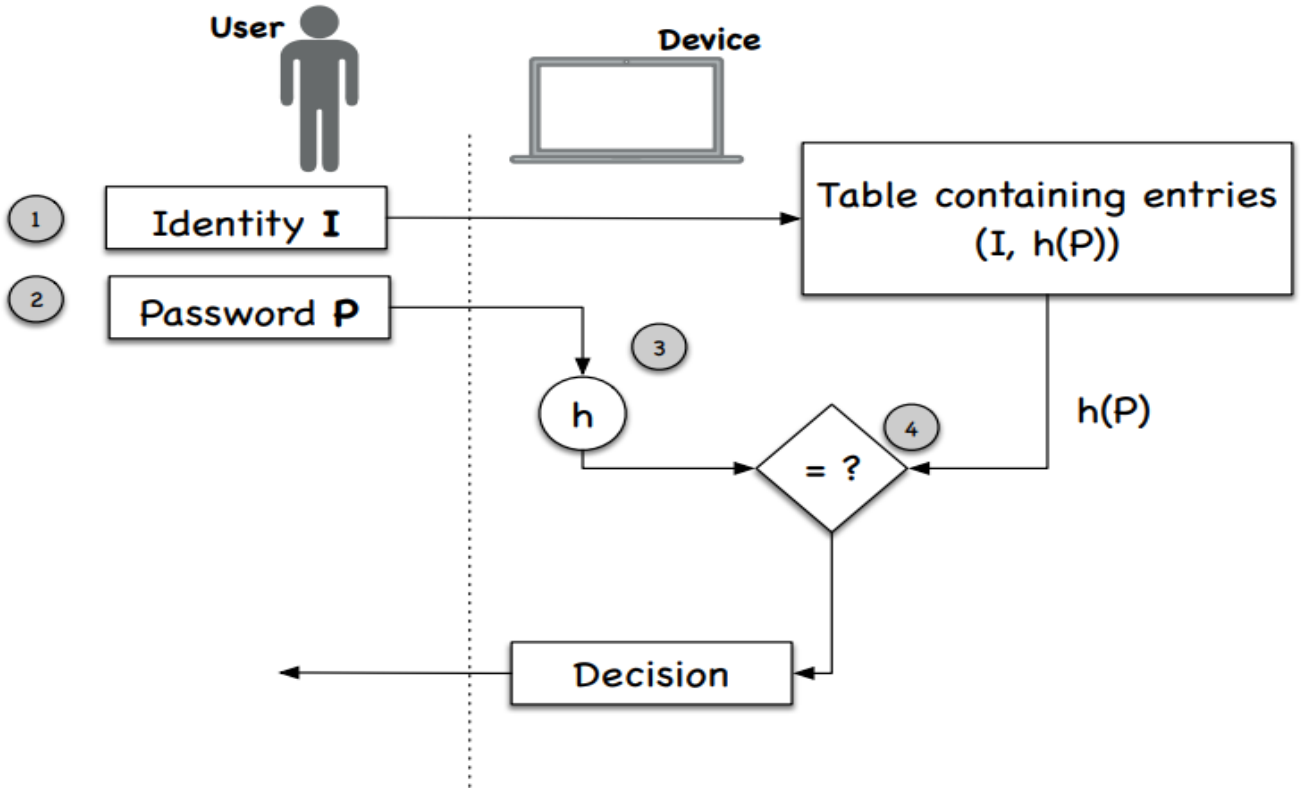
If your seed labs ZIP doesn't match that that hash, then you might have a modified OS image

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

Two people that have the same password will have the **same hash** → not good!

Salt is just some random string appended to a password

`WdRrWCQzpassword123`

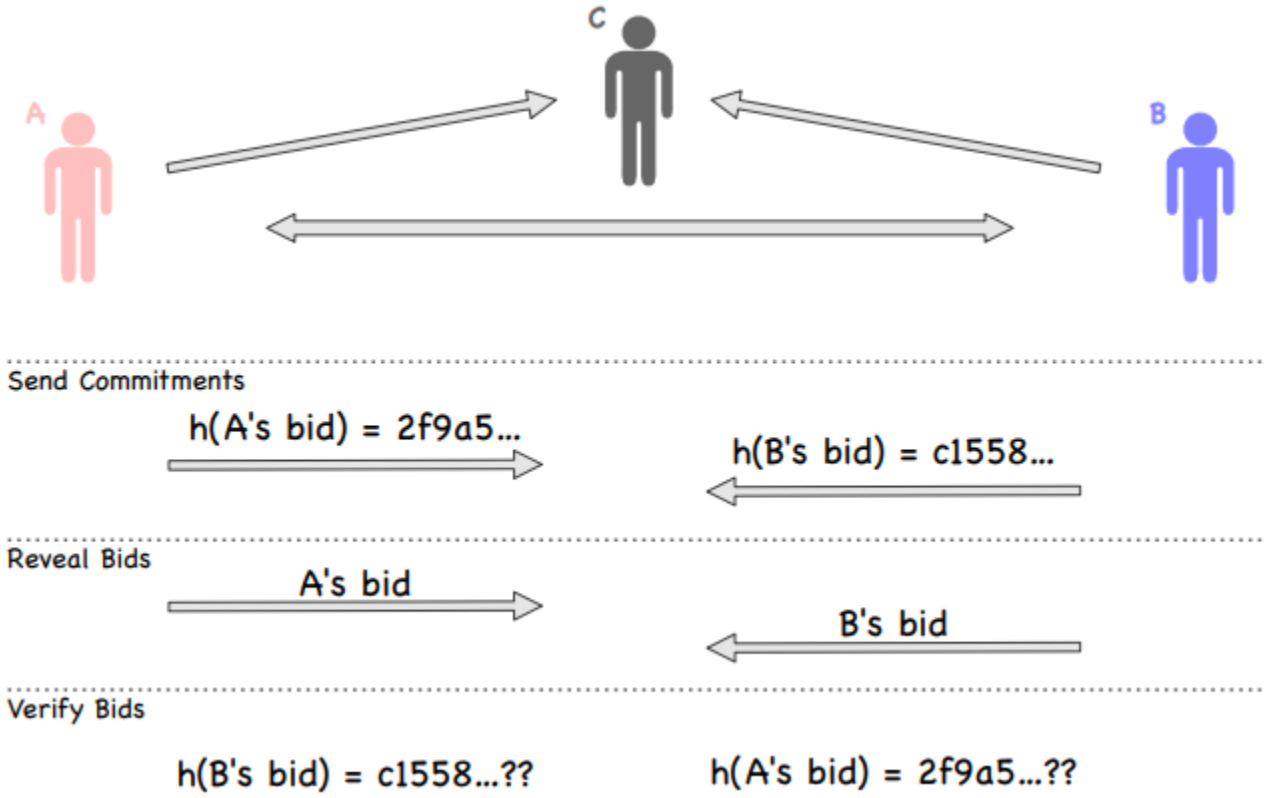
When a service uses salted passwords, the same input (password) can result in different hashes! → good

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

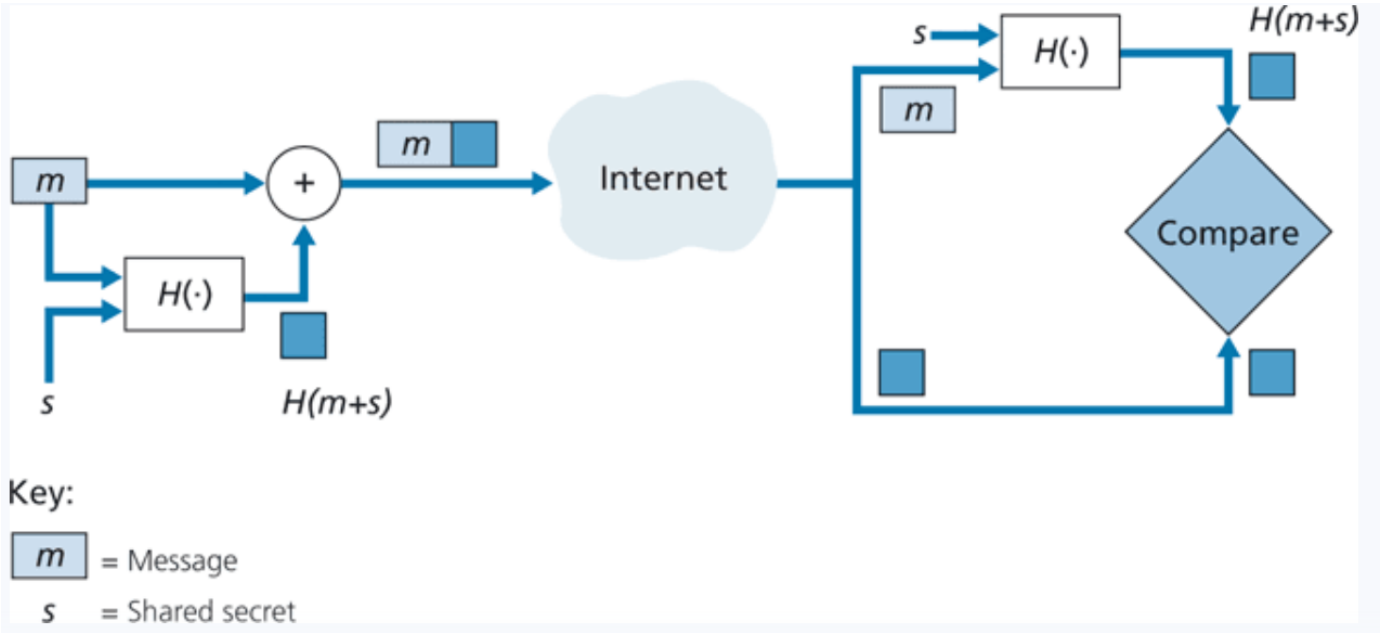
- Example: Fair Games



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



Collision Attacks

```
hello_world
```

89defae676abd3e3a42b41df17c40096

```
hello_world
```

89defae676abd3e3a42b41df17c40096

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

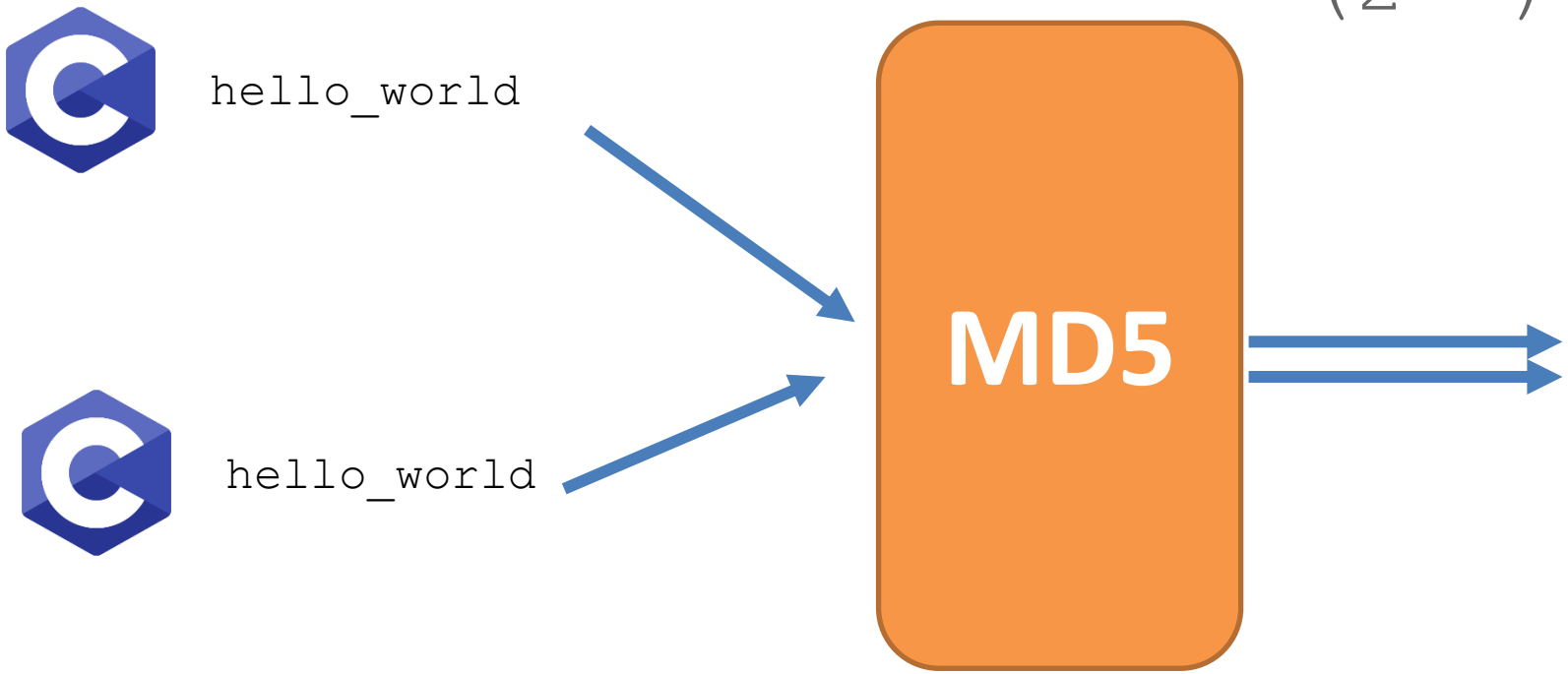
Hash Collision Attacks compromise the integrity of a program by creating a malicious file that has a same hash

Collision Attacks

00000000000000000000000000000000
00000000000000000000000000000001
00000000000000000000000000000010

There is a very large amount of possible hashes

$\sim (2^{128})$



...

106a7d06be131315e25a7cbe57af398c

...

How likely is? Very unlikely?

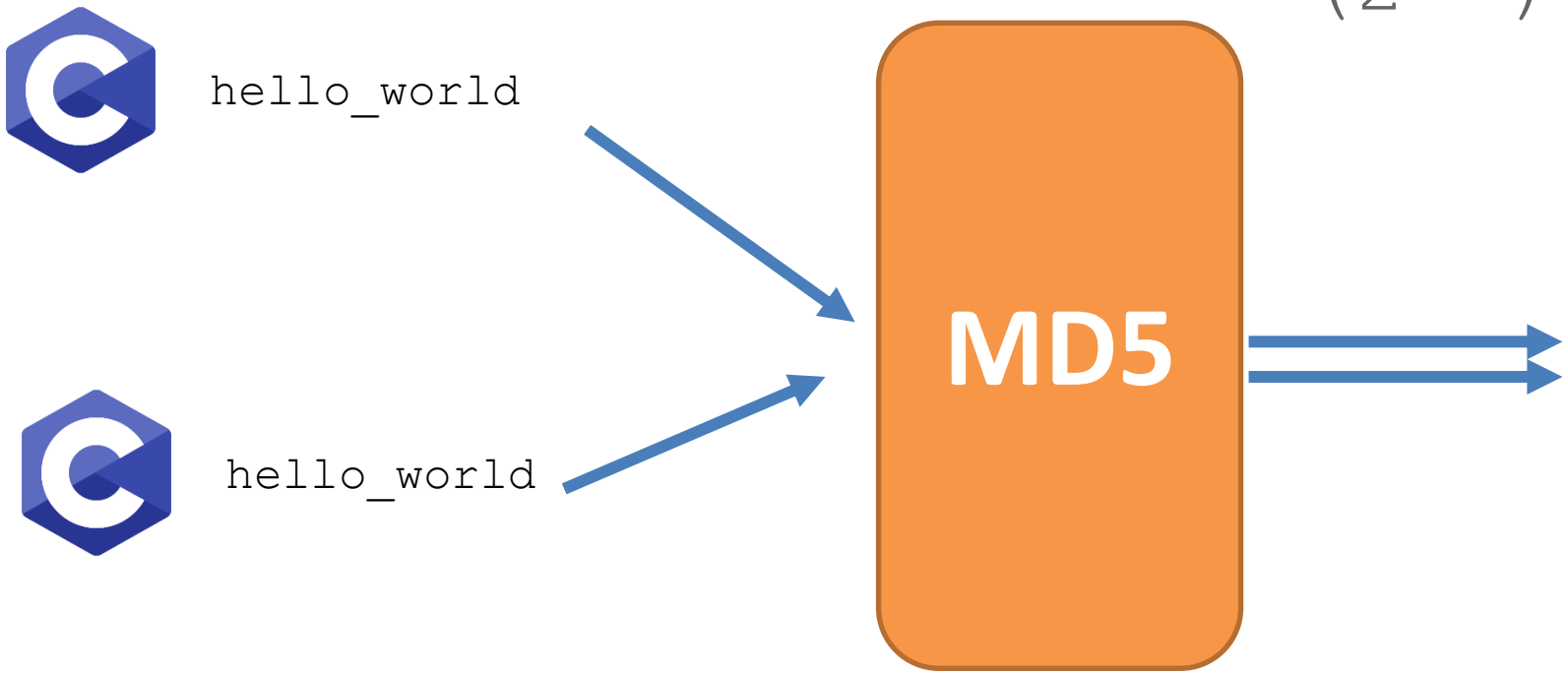
EEFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
EFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

Collision Attacks

00000000000000000000000000000000
00000000000000000000000000000001
00000000000000000000000000000010

There is a very large amount of possible hashes

$\sim (2^{128})$



…

106a7d06be131315e25a7cbe57af398c

…

EEFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
EFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

How likely is? Very unlikely?

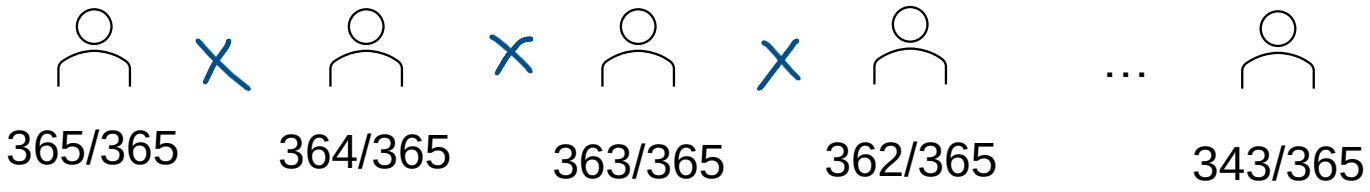
More likely than you think…

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

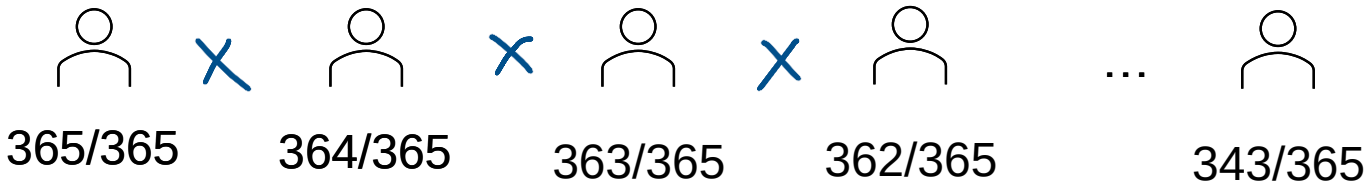


Birthday Paradox

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

$1 - .4927$

$= \approx 50\%$

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

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

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…

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

Motivation

Forging public-key certificates

md5 (

) == md5 (

)

- 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!

Integrity of Programs

md5 (

) == md5 (

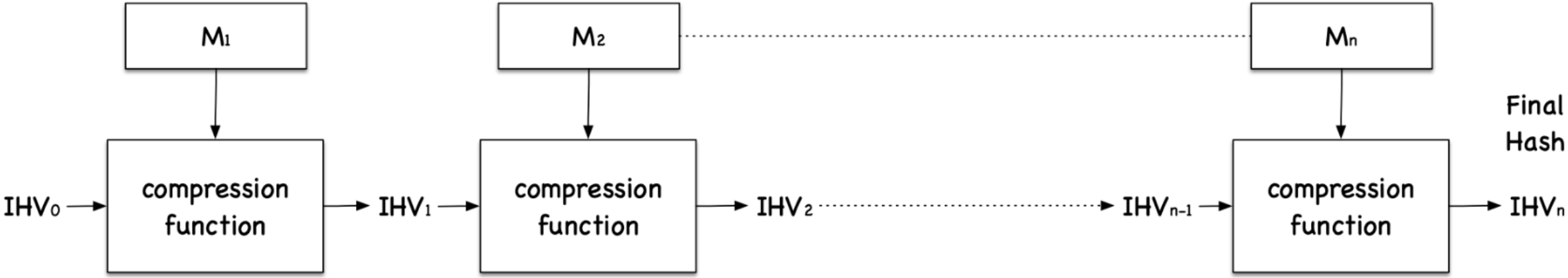
)

- 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

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 <u>64 bytes</u>

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)

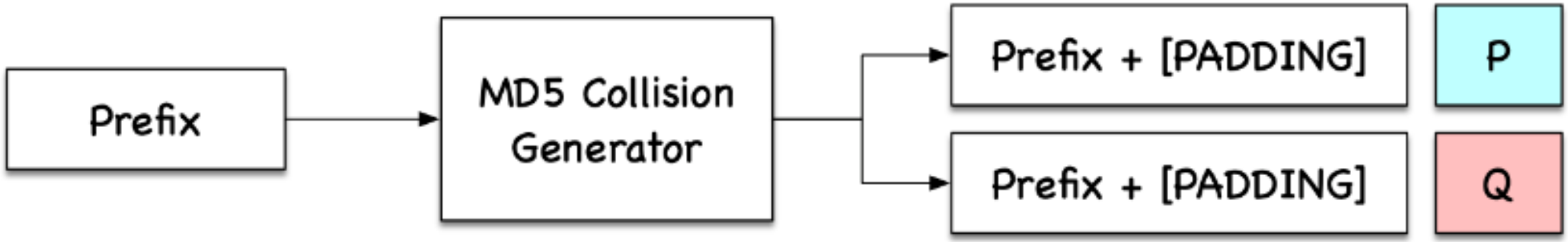
On our VM, we have a tool called **md5collgen** that will generate two files with the **same prefix** ← We get to choose this prefix!

```
[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)

What if out prefix is a multiple of 64?



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
```

```
[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: 4748 4a4a 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  ?.:...K.Y..9j*.m
00000060: dfc8 b3c7 7cc5 0d68 0b02 d253 b15d d615  ....|..h...S.]..
00000070: ff21 3cc5 c79b 666c 9f66 e355 8678 44c0  .!<...fl.f.U.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 a134 d2ee  .#&Y....h*M..4..
000000b0: ff15 ba1c d348 e152 11ae 7d5a 3557 47d1  .....H.R..}Z5WG.
[11/17/22]seed@VM:~/.../07_hash$ xxd out2.bin
00000000: 6162 6364 6566 6768 696a 6b6c 6d6e 6f70  abcdefghijklmnop
00000010: 7172 7374 7576 7778 797a 4142 4344 4546  qrstuvwxyzABCDEF
00000020: 4748 4a4a 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$
```

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