CSCI 476: Computer Security

Hashing (Part 1)

Reese Pearsall

Fall 2023

Announcements

Lab 8 due Sunday December 3rd

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

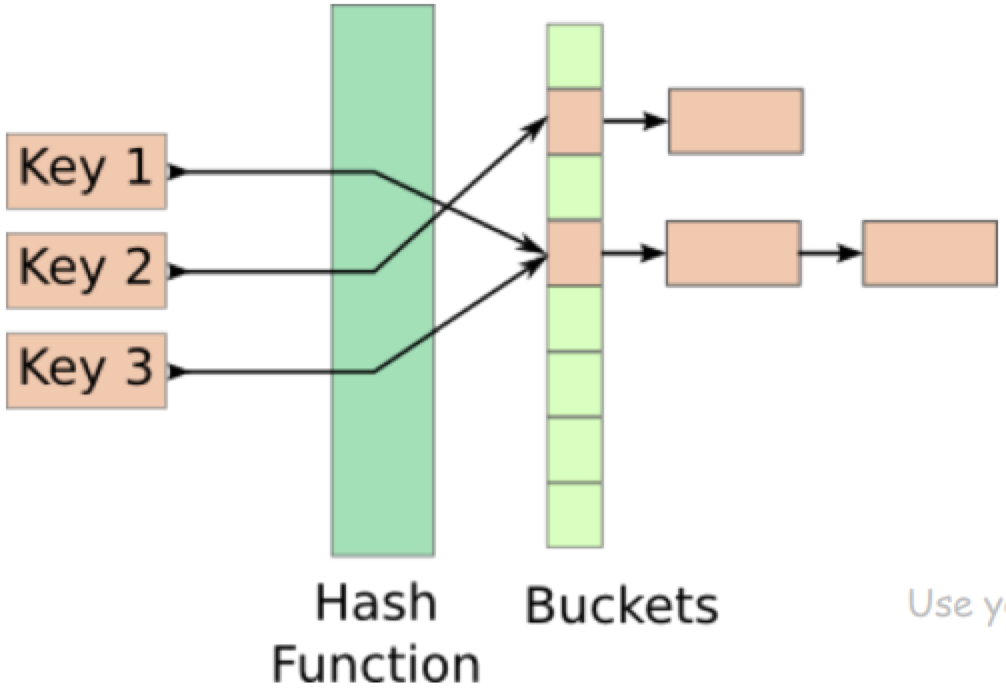
Lab 8

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**



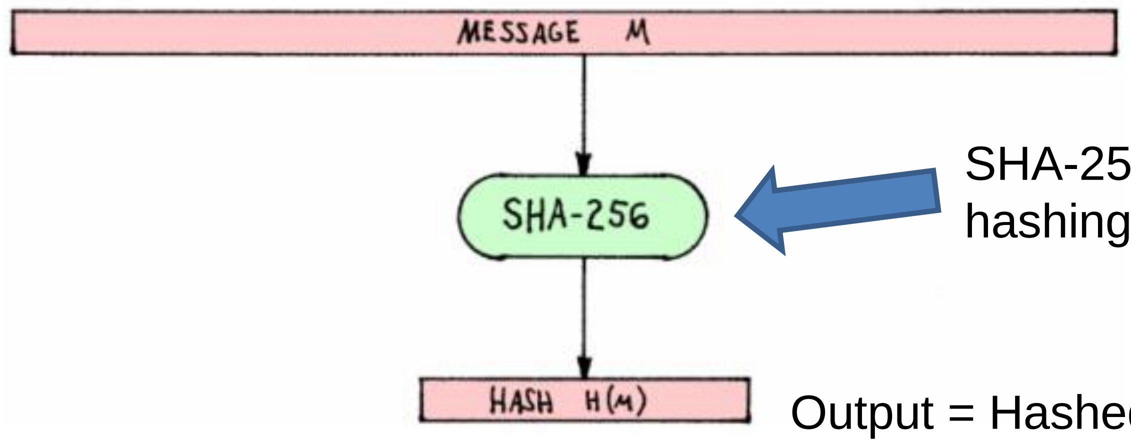
How many buckets?
What to do if two keys map to the same bucket?

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



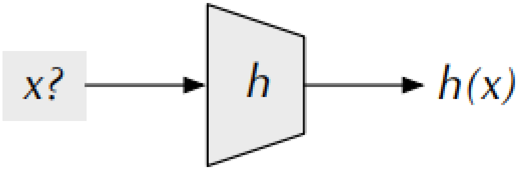
Input = Message M

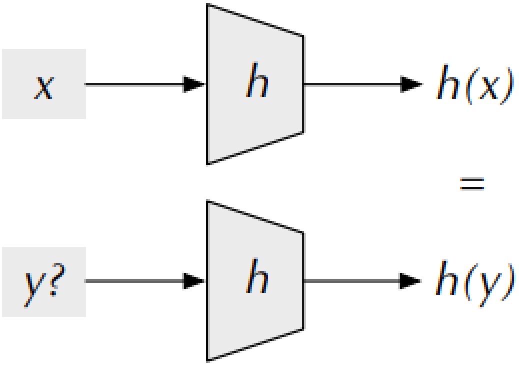
SHA-256 = A cryptographically secure hashing function

Output = Hashed Message M

Hash Functions Properties

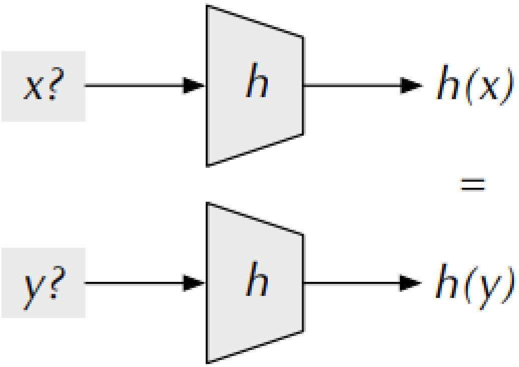
- **Preimage Resistance ("One-Way")**
  Given h(x) = z, hard to find x
  (or any input that hashes to z for that matter)



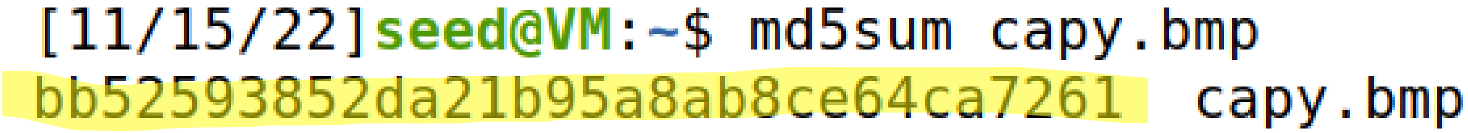


- **Second Preimage Resistance**
  Given x and h(x), hard to find y s.t. h(x) = h(y)

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



Hash Functions Properties (tl;dr)



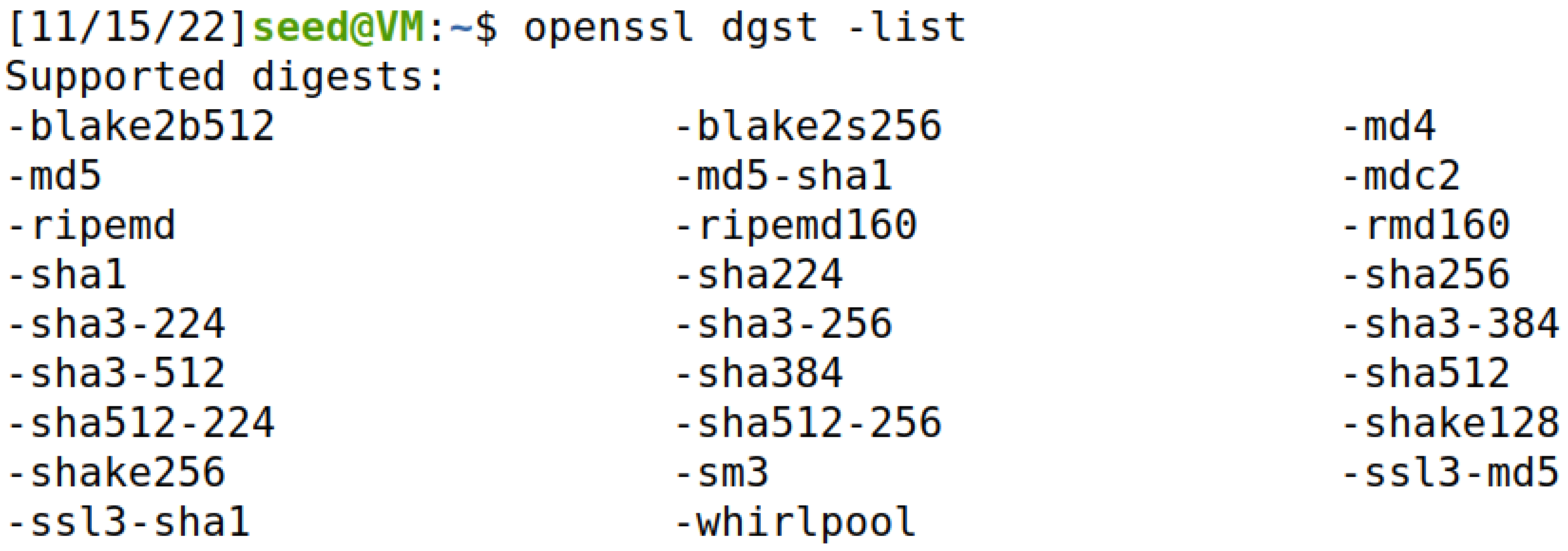
```
[11/15/22]seed@VM:~$ md5sum capy.bmp
bb52593852da21b95a8ab8ce64ca7261  capy.bmp
```

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

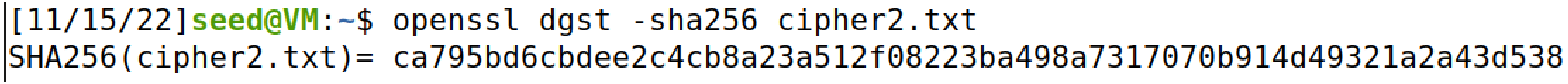
The go-to way to reverse a hash is through brute force

Computing Hashes with OpenSSL



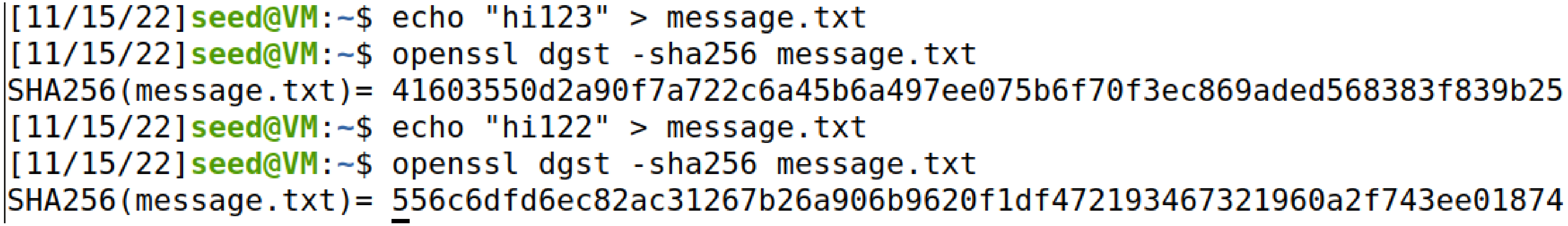
```
[11/15/22]seed@VM:~$ openssl dgst -list
Supported digests:
-blake2b512                -blake2s256                -md4
-md5                       -md5-sha1                  -mdc2
-ripemd                    -ripemd160                 -rmd160
-sha1                      -sha224                    -sha256
-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



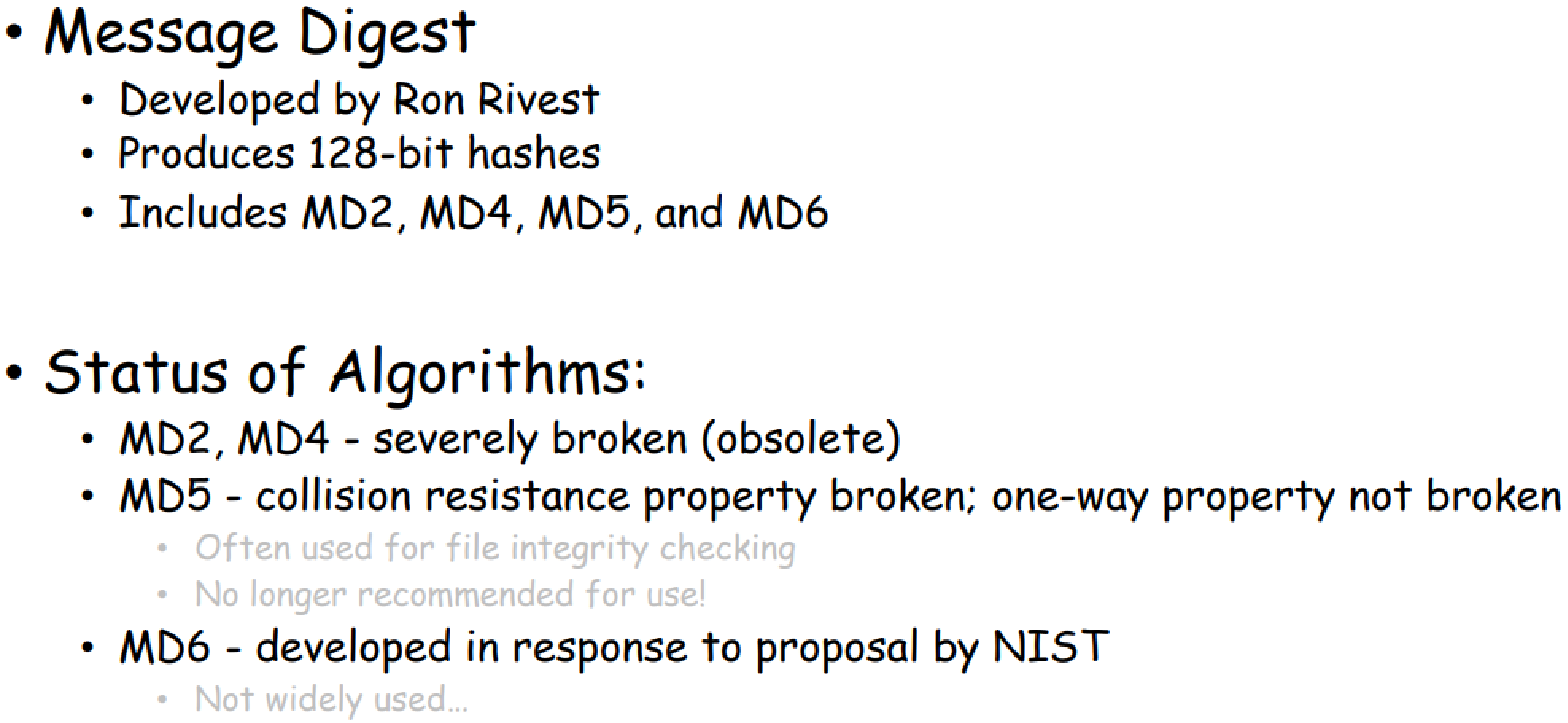
```
[11/15/22]seed@VM:~$ openssl dgst -sha256 cipher2.txt
SHA256(cipher2.txt)= ca795bd6cbdee2c4cb8a23a512f08223ba498a7317070b914d49321a2a43d538
```

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



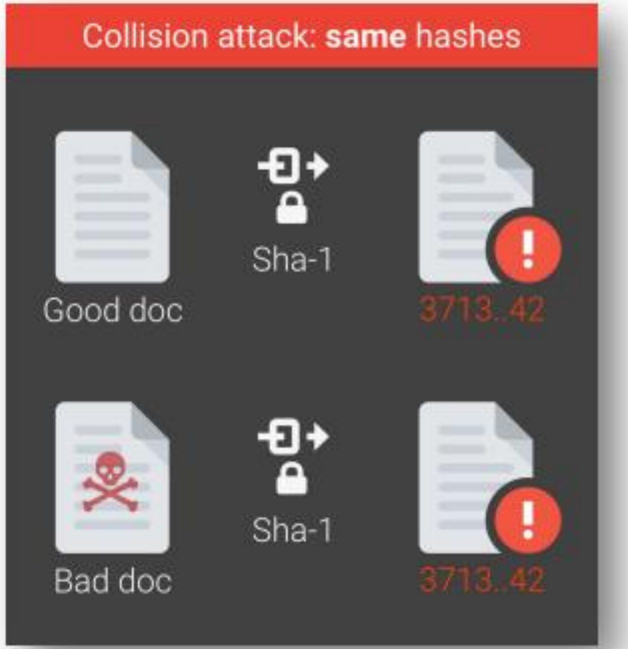
- Message Digest
  - Developed by Ron Rivest
  - Produces 128-bit hashes
  - Includes MD2, MD4, MD5, and MD6
- Status of Algorithms:
  - MD2, MD4 - severely broken (obsolete)
  - MD5 - collision resistance property broken; one-way property not broken
    - Often used for file integrity checking
    -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

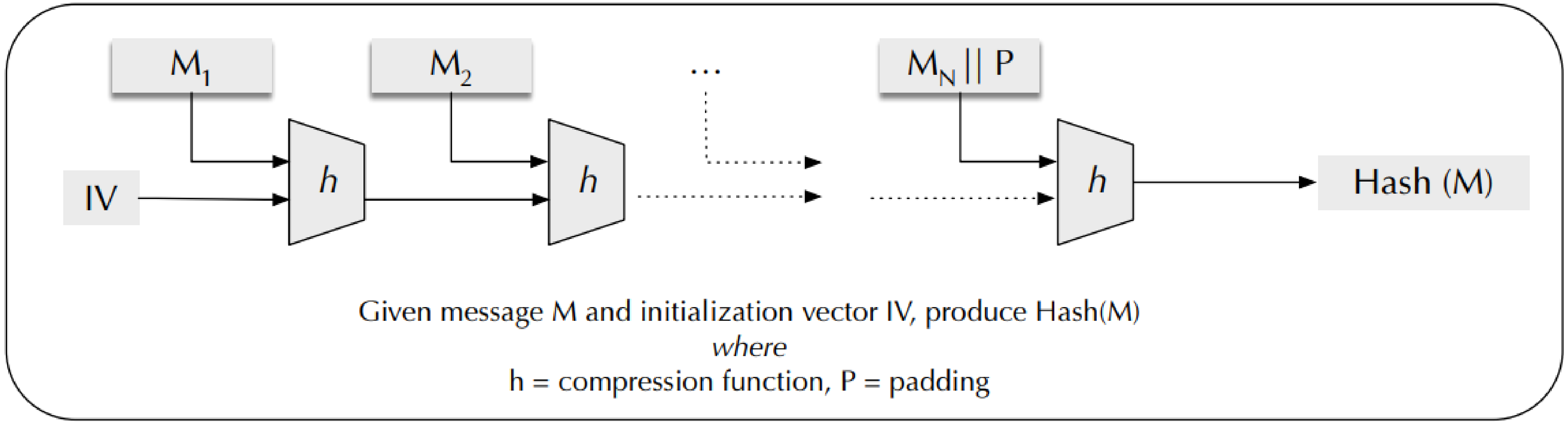
- Status of Algorithms:
  - SHA-0: withdrawn due to flaw
  - SHA-1: Designed by NSA   Collision attack found in 2017
  - SHA-2: Designed by NSA
    - Includes SHA-256 and SHA-512 + other truncated versions;
    - No significant attack found yet…
  - SHA-3: Not Designed by NSA
    - Released in 2015;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

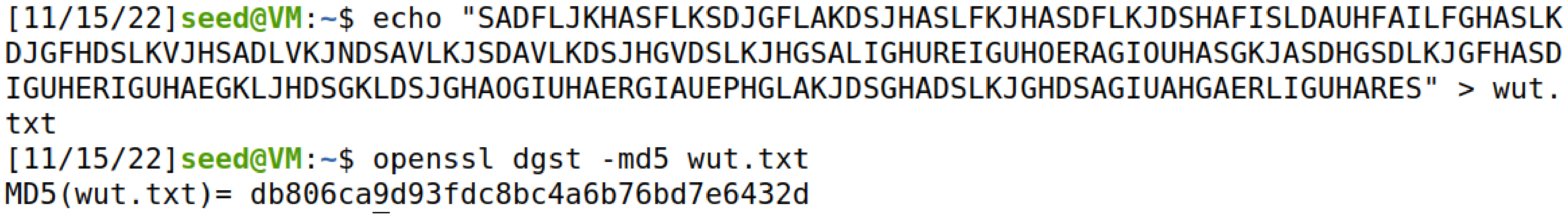
Most hash algorithms (e.g., MD5, SHA-1, SHA-2) use a <u>Merkle-Damgard</u> construction:



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

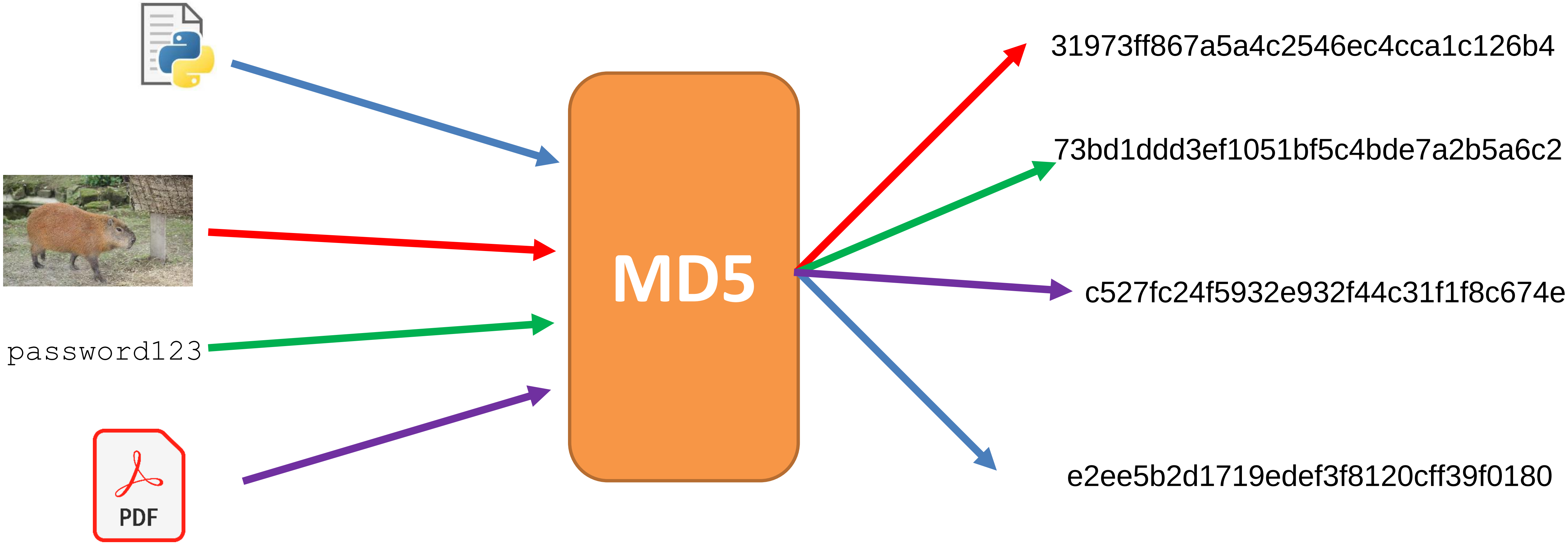
Calculating Hashes in Programming Languages

```
# 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)



password123

MD5

31973ff867a5a4c2546ec4cca1c126b4

73bd1ddd3ef1051bf5c4bde7a2b5a6c2

c527fc24f5932e932f44c31f1f8c674e

e2ee5b2d1719edef3f8120cff39f0180

FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

What are some uses for hashing?

Applications of Hashing

Integrity Verification

A CSCI 112 Student

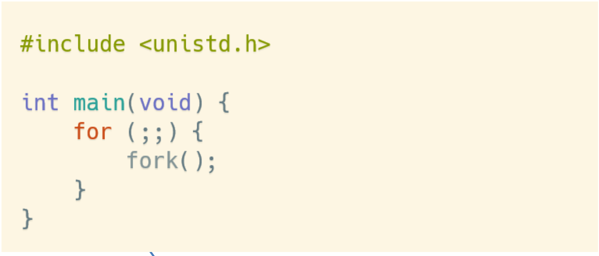
`hello_world`

Applications of Hashing

Integrity Verification

A CSCI 112 Student

hello_world



```
#include <unistd.h>

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

hello_world

What if the message got tampered with?

Instructor

hello_world

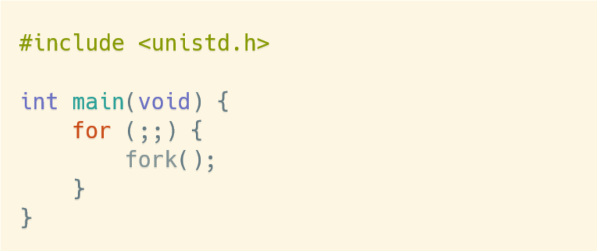
She will have no idea because this executable program seems totally normal and came from a trustworthy source

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

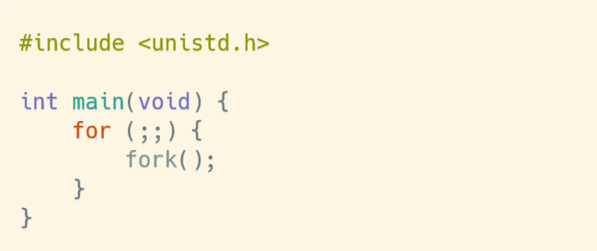
Applications of Hashing

Integrity Verification

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

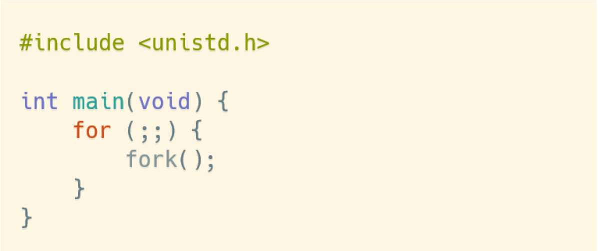
Applications of Hashing

Integrity Verification

hello_world

89defae676abd3e3a42b41df17c40096

A CSCI 112 Student



```
#include <unistd.h>

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

hello_world

What if the message got tampered with?

hello_world

b0608c4e1775ad8f92e7b5c191774c5d

Instructor

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

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!

Applications of Hashing

Integrity Verification

A CSCI 112 Student

hello_world

89defae676abd3e3a42b41df17c40096

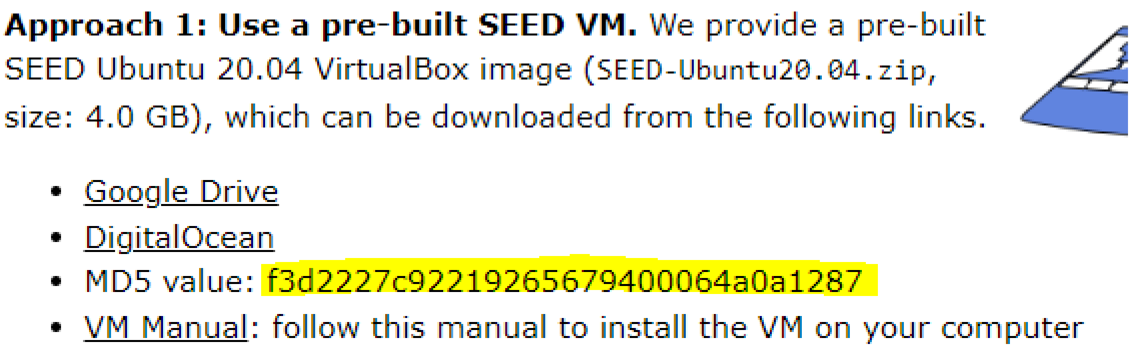
Instructor

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

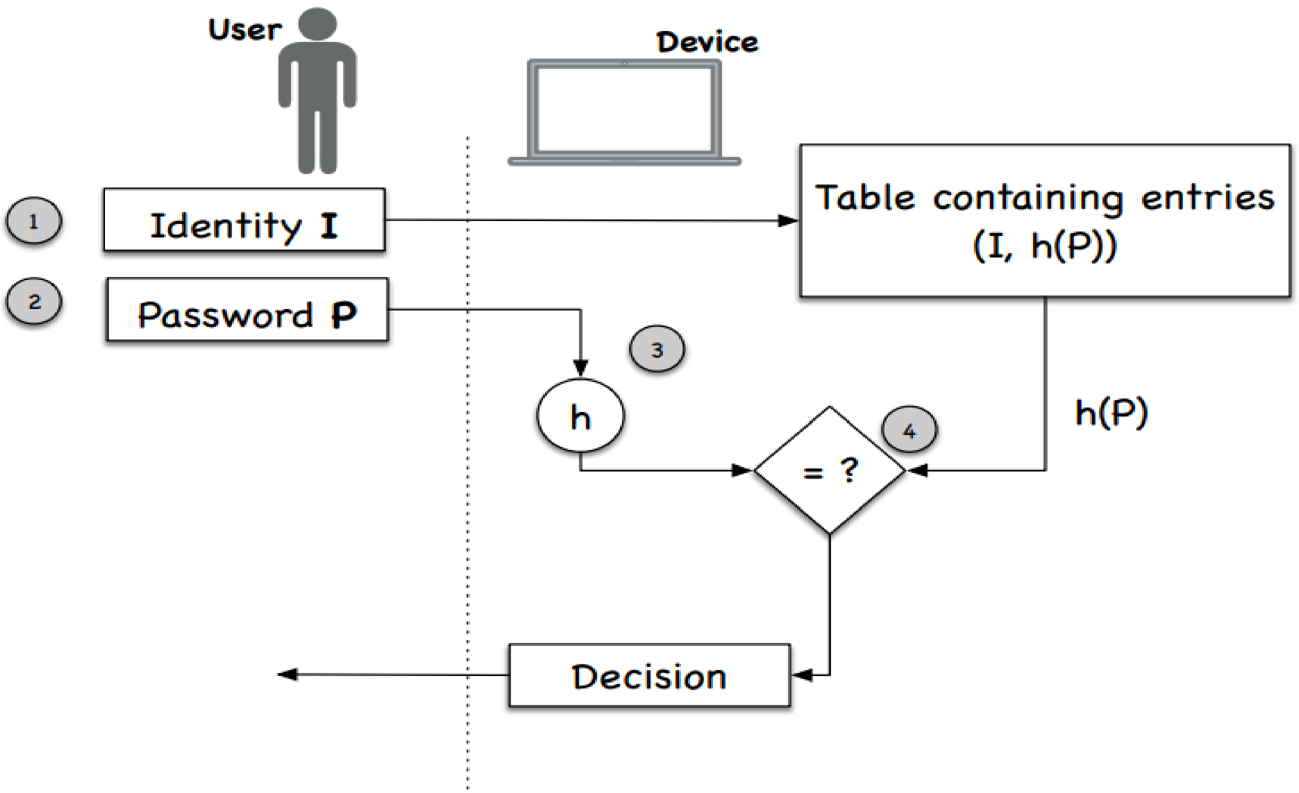
If your seed labs ZIP doesn't match that that hash, then you might have a modified OS image

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

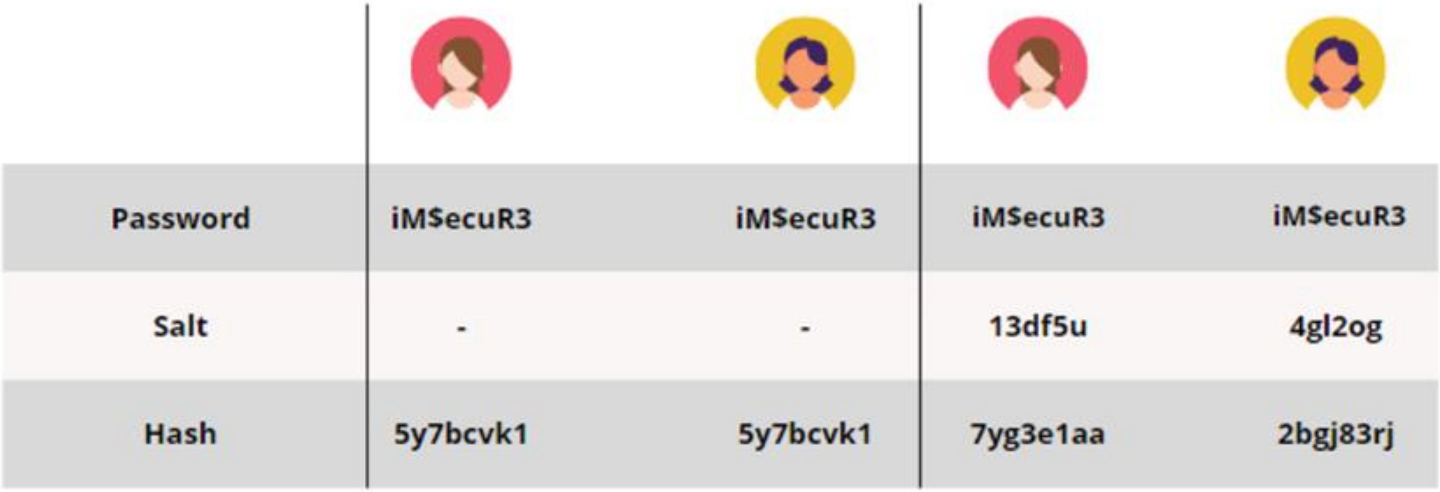
Password Verification

Two people that have the same password will have the same hash **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

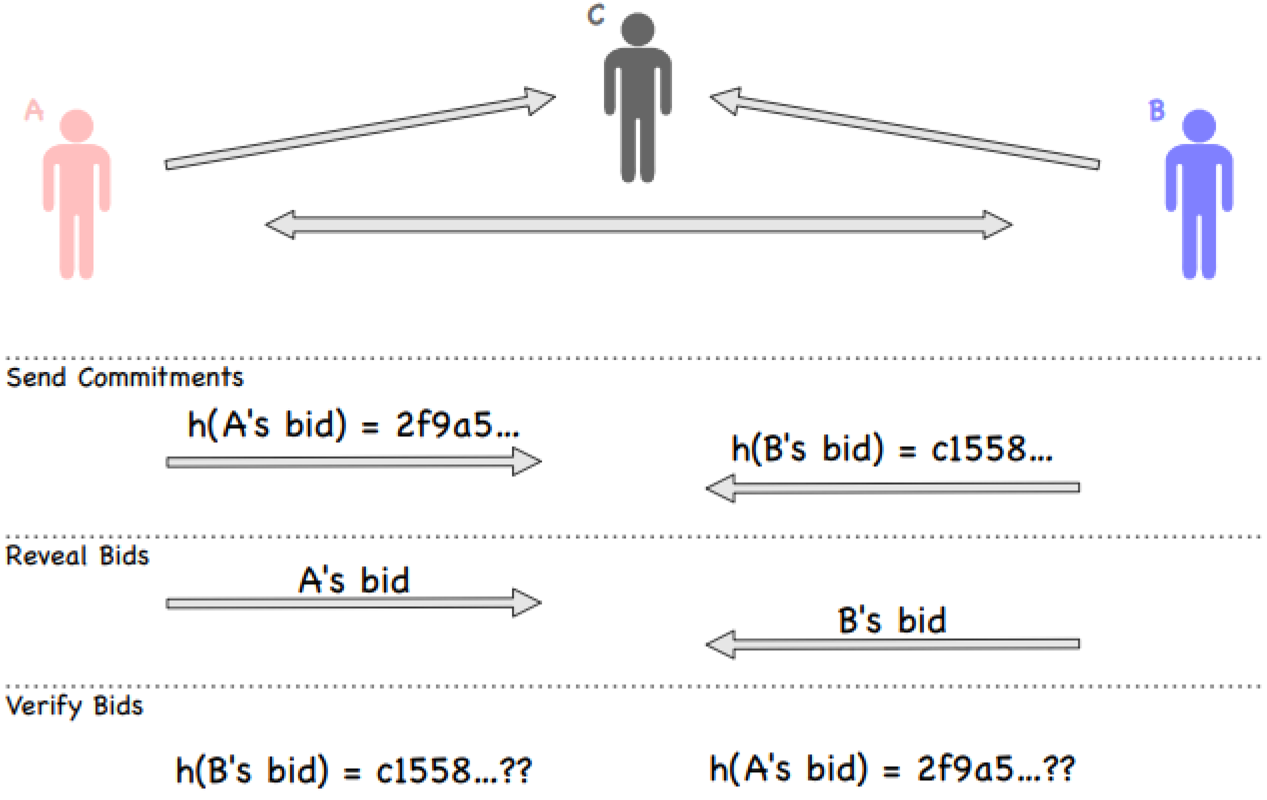


Password	iM$ecuR3	iM$ecuR3	iM$ecuR3	iM$ecuR3
Salt	-	-	13df5u	4gl2og
Hash	5y7bcvk1	5y7bcvk1	7yg3e1aa	2bgj83rj

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



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...??

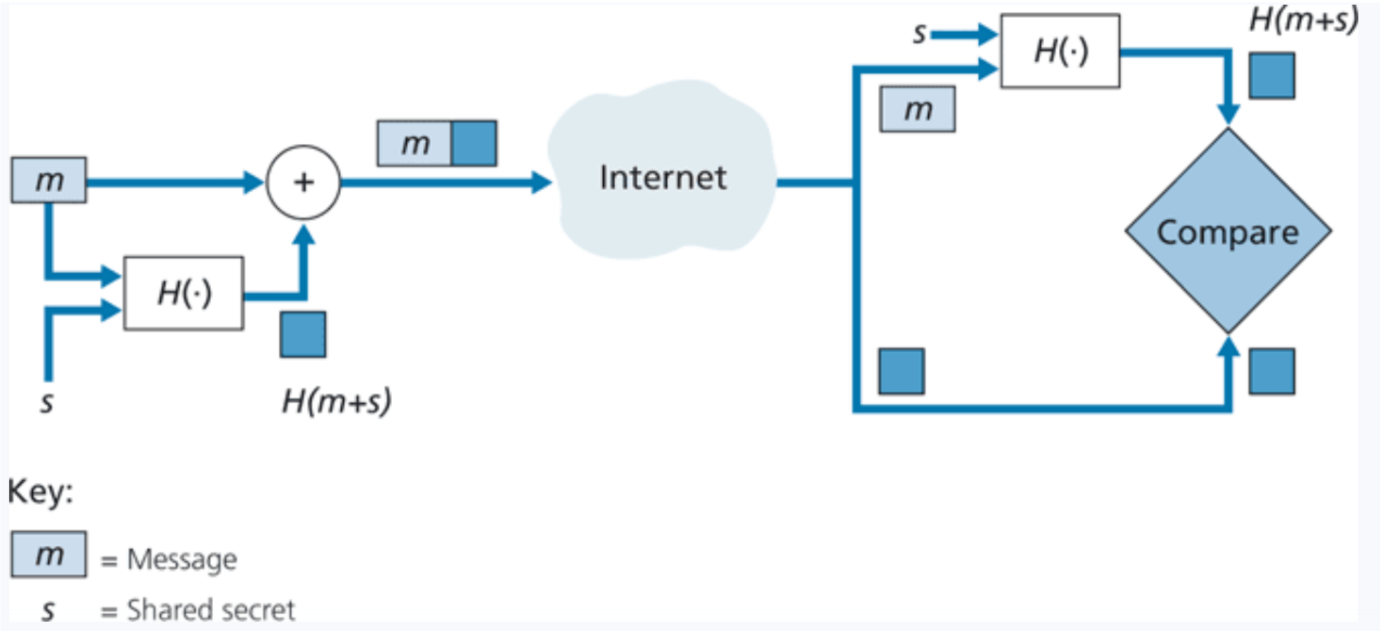
h(A's bid) = 2f9a5...??

Applications of 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



Attacks on Hashing

Suppose we get a hash for an unsalted password

```
cc3a0280e4fc1415930899896574e118
```

What could we do to retrieve the original password?

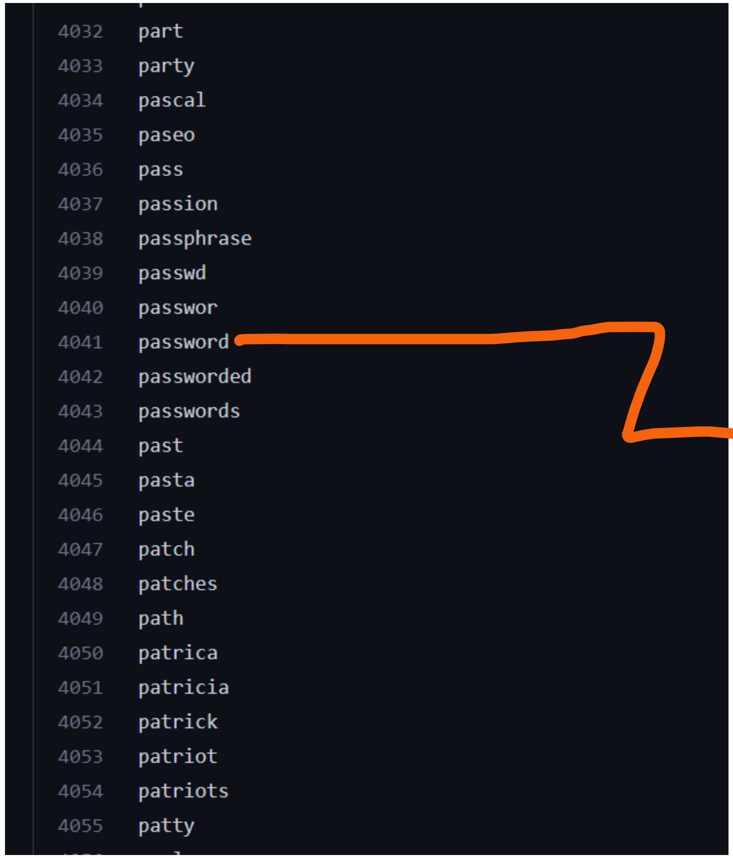
- **Brute Force**

❑ Dictionary Attack
❑ Rainbow Tables

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

Dictionary Attack

We will use an existing list of common passwords



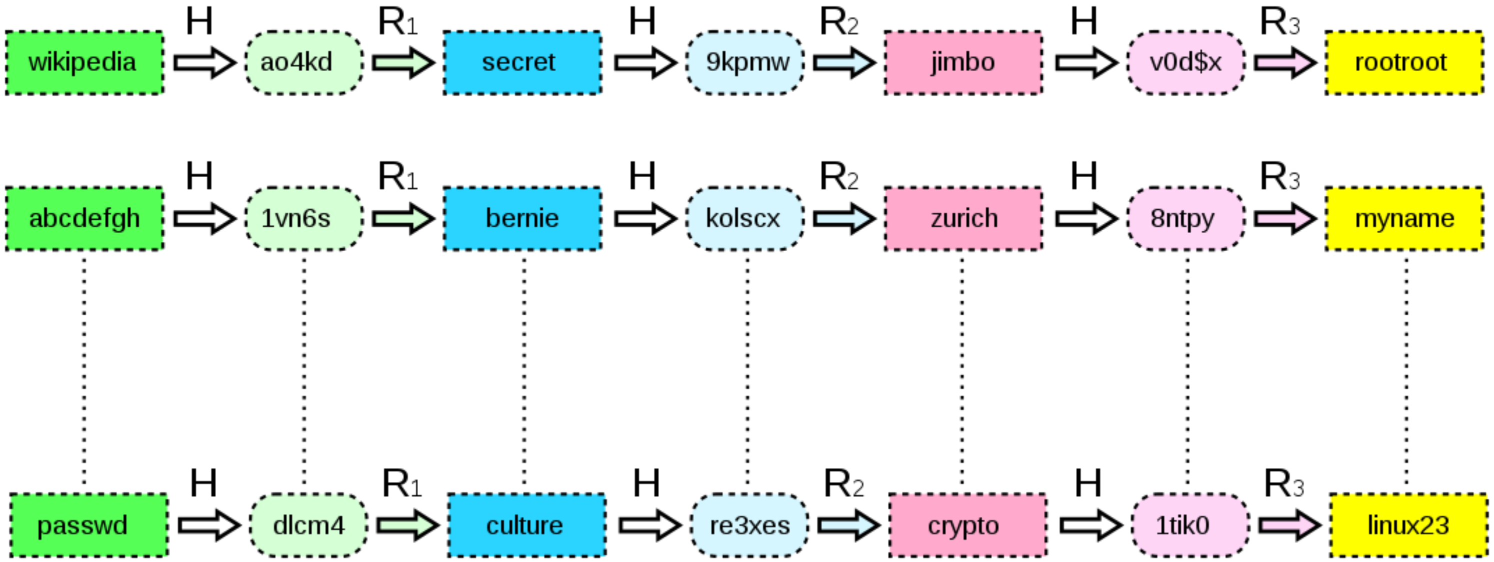
```
part
party
pascal
paseo
pass
passion
passphrase
passwd
passwor
password
passworded
passwords
past
pasta
paste
patch
patches
path
patrica
patricia
patrick
patriot
patriots
patty
```

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

MD5 = ? =

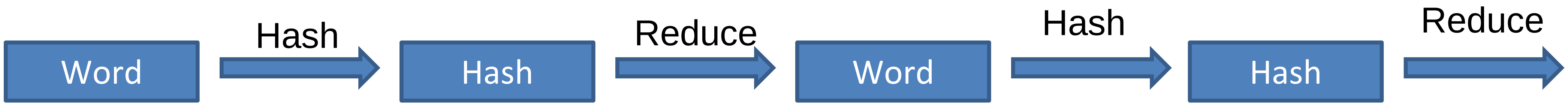
This works for cracking weak, unsalted passwords

Rainbow Tables


wikipedia H ao4kd R1 secret H 9kpmw R2 jimbo H v0d$x R3 rootroot
abcdefgh H 1vn6s R1 bernie H kolscx R2 zurich H 8ntpy R3 myname
passwd H dlcm4 R1 culture H re3xes R2 crypto H 1tik0 R3 linux23

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

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



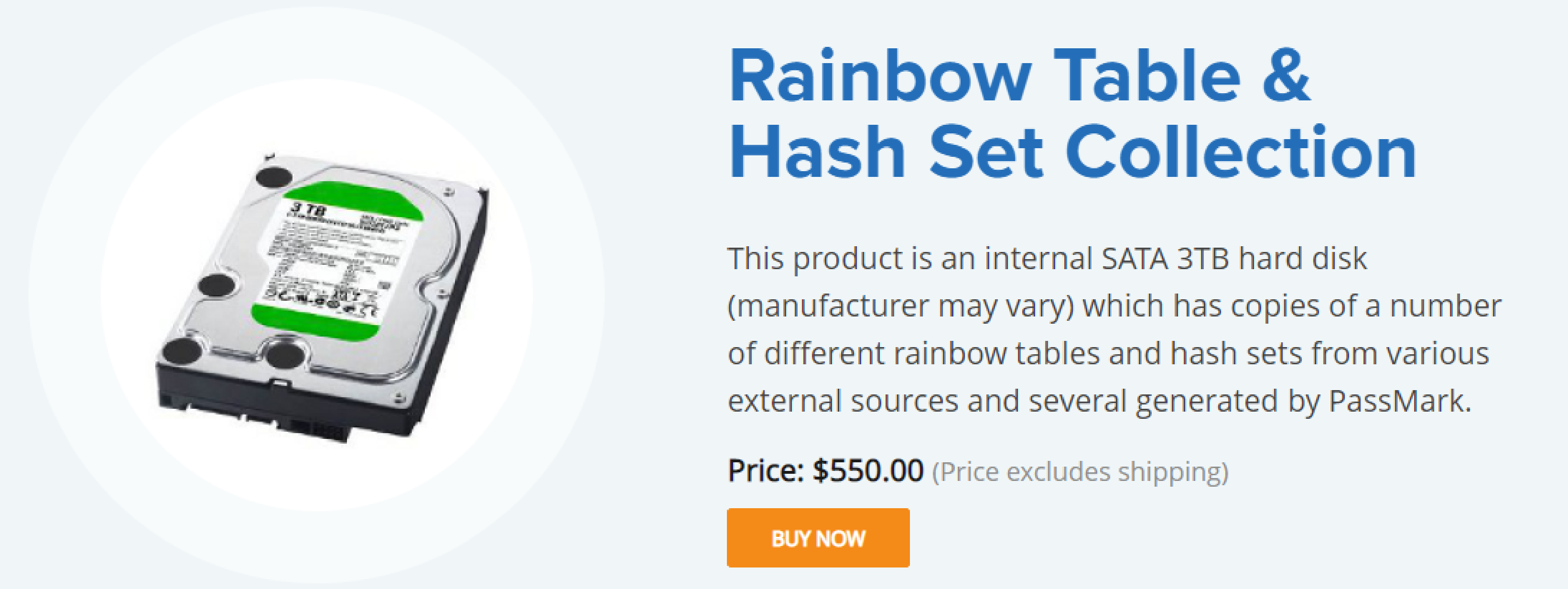
Looking up a value in the rainbow table can happen quick, but these files are typically very large

Not efficient for complex, salted passwords

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

Project-RainbowCrack

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)

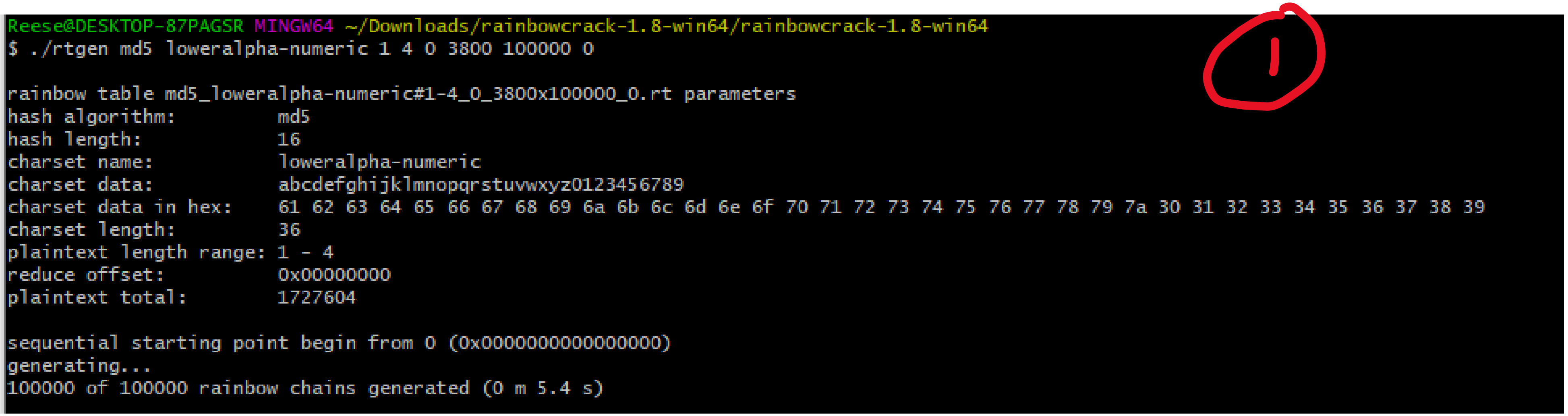
Tables for alphanumeric, special character passwords can take a long time to generate, so instead of doing it yourself, you can buy rainbow tables that other people have generated!

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

- Project-RainbowCrack

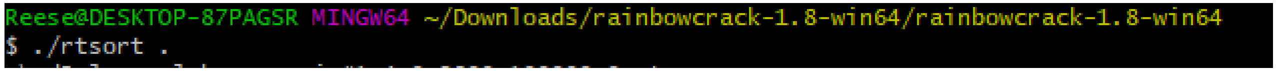
Rainbow Tables using RainbowCrack

1



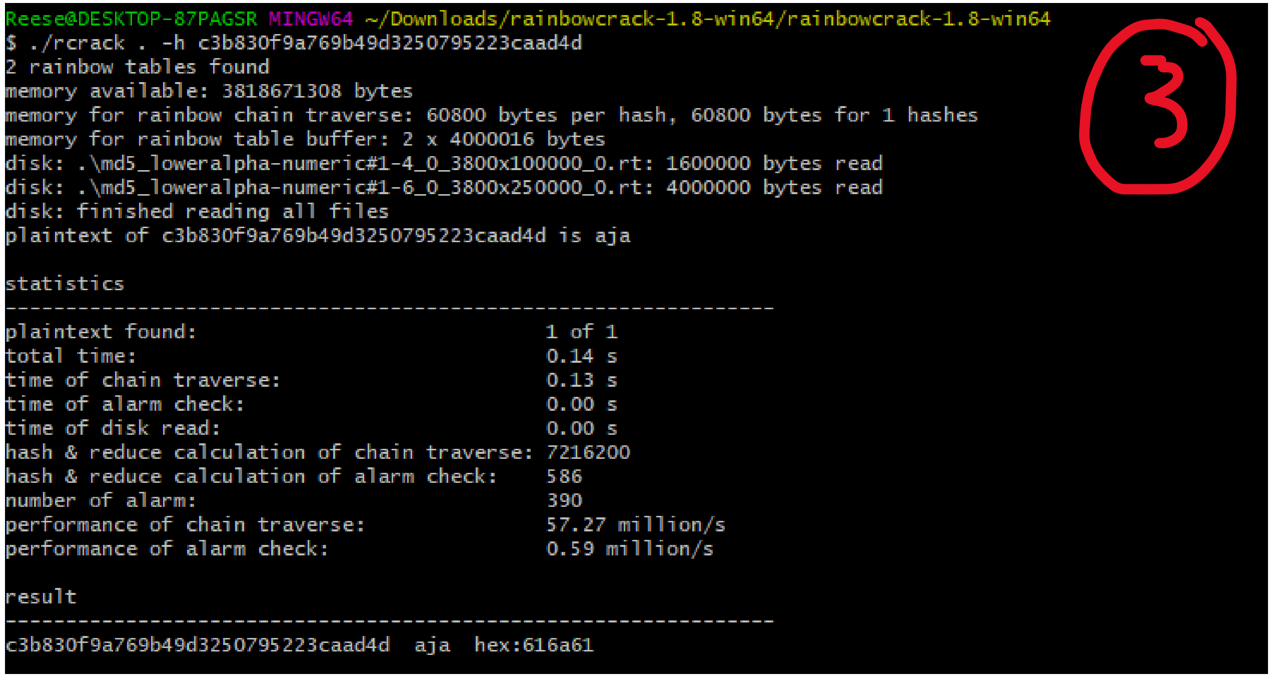
```
Reese@DESKTOP-87PAGSR MINGW64 ~/Downloads/rainbowcrack-1.8-win64/rainbowcrack-1.8-win64
$ ./rtgen md5 loweralpha-numeric 1 4 0 3800 100000 0

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
```

Collision Attacks

`hello_world`

89defae676abd3e3a42b41df17c40096

`hello_world`

89defae676abd3e3a42b41df17c40096

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

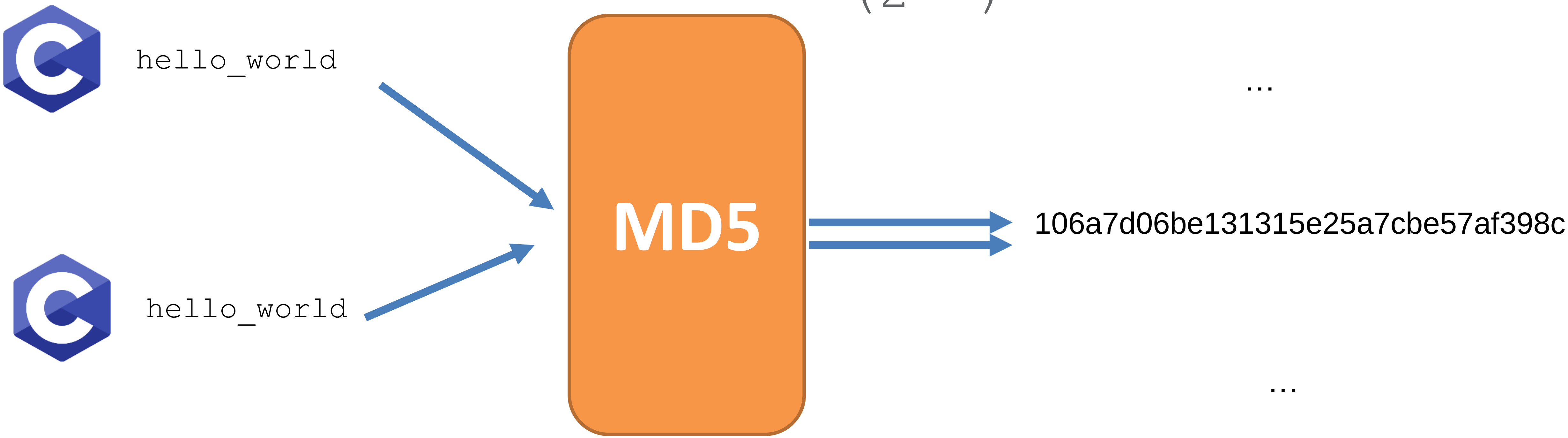
Hash Collision Attacks compromise the integrity of a program by creating a malicious file that has a same hash

Collision Attacks

00000000000000000000000000000000
00000000000000000000000000000001
00000000000000000000000000000010

There is a very large amount of possible hashes

$\sim (2^{128})$



hello_world

hello_world

MD5

106a7d06be131315e25a7cbe57af398c

...

...

How likely is? Very unlikely?

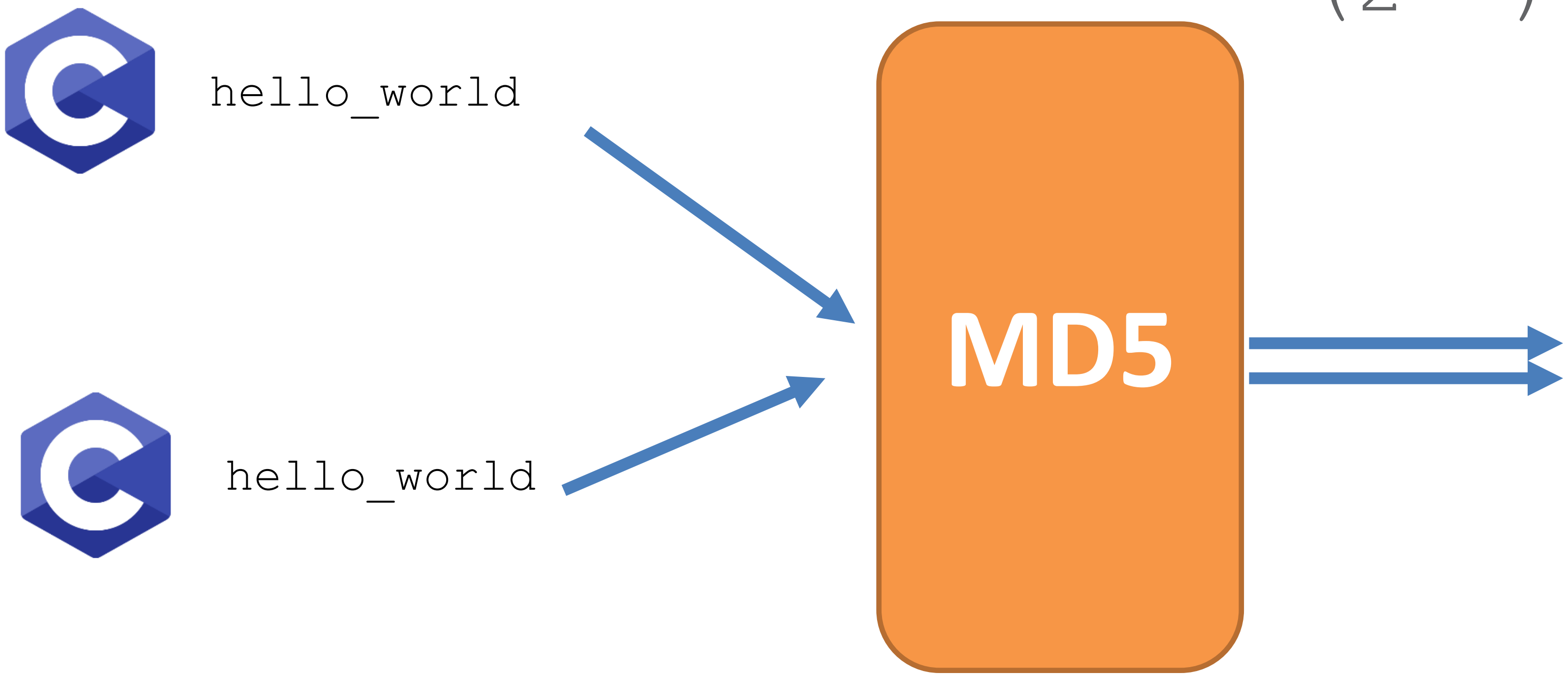
EEFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
EFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

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

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

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

$$1 - .4927$$

$$= \approx 50\%$$

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

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

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…