WEEK 4

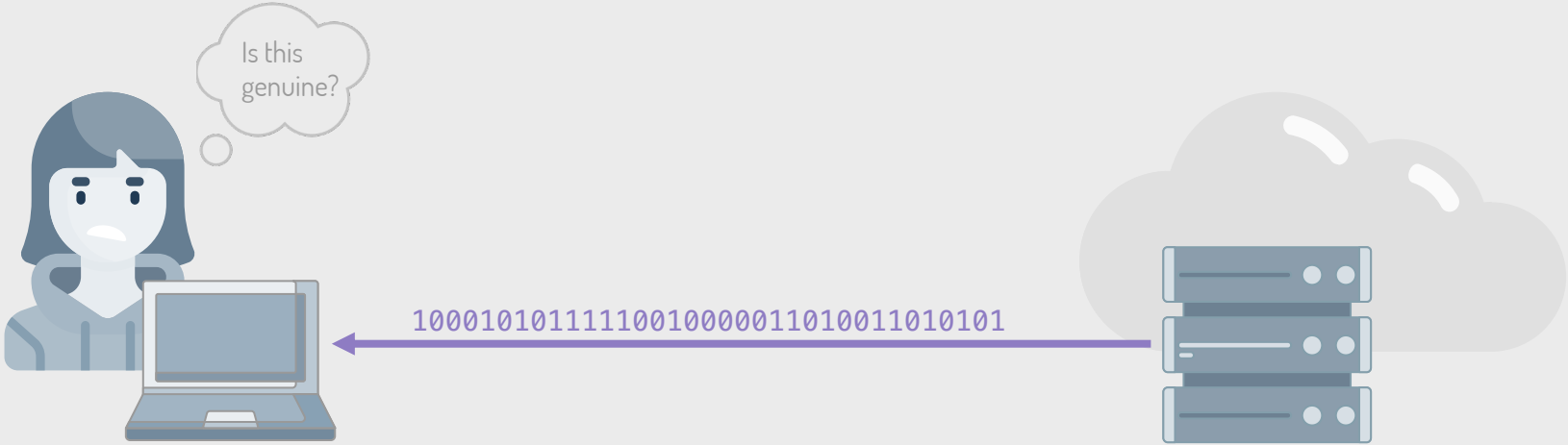
FINGERPRINTING WITH HASH FUNCTIONS

SE 4472 - Information Security

Western Engineering

WHISPER LAB

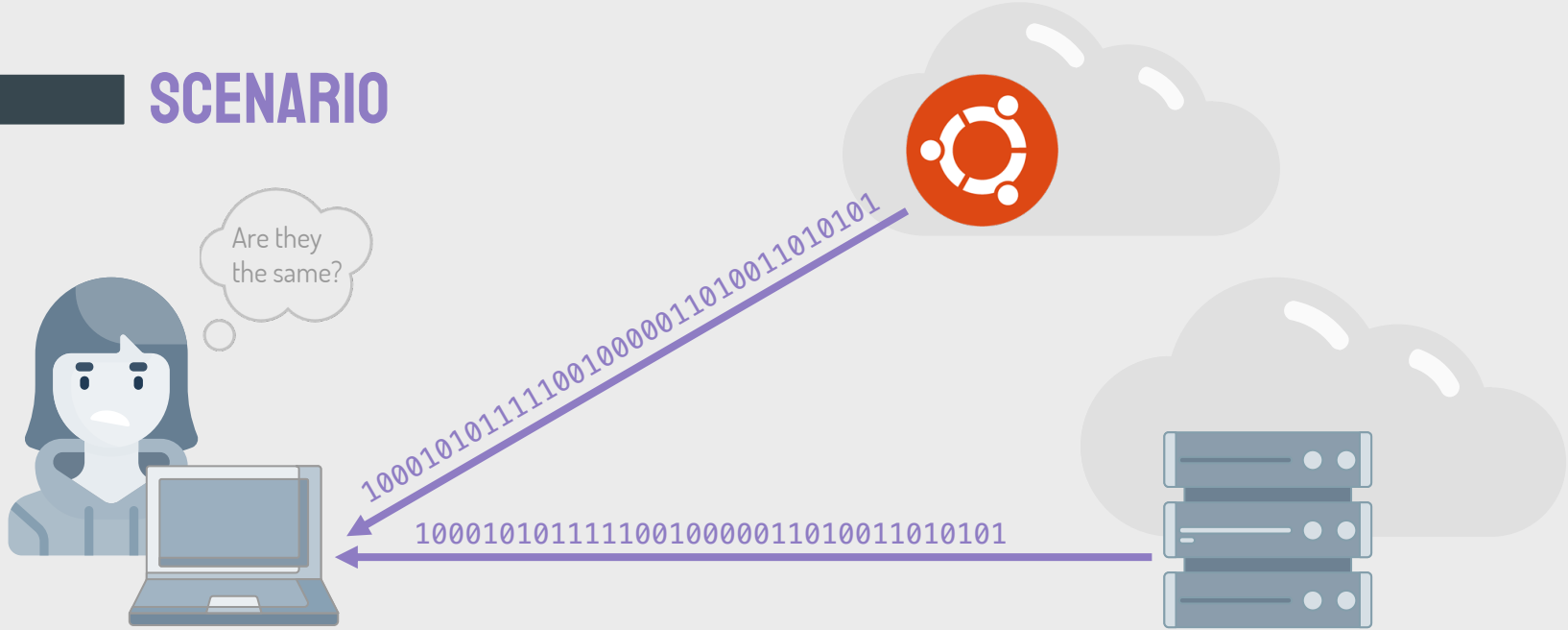
SCENARIO



You download the latest Ubuntu release from a cloud server

How do you know it's not malware?

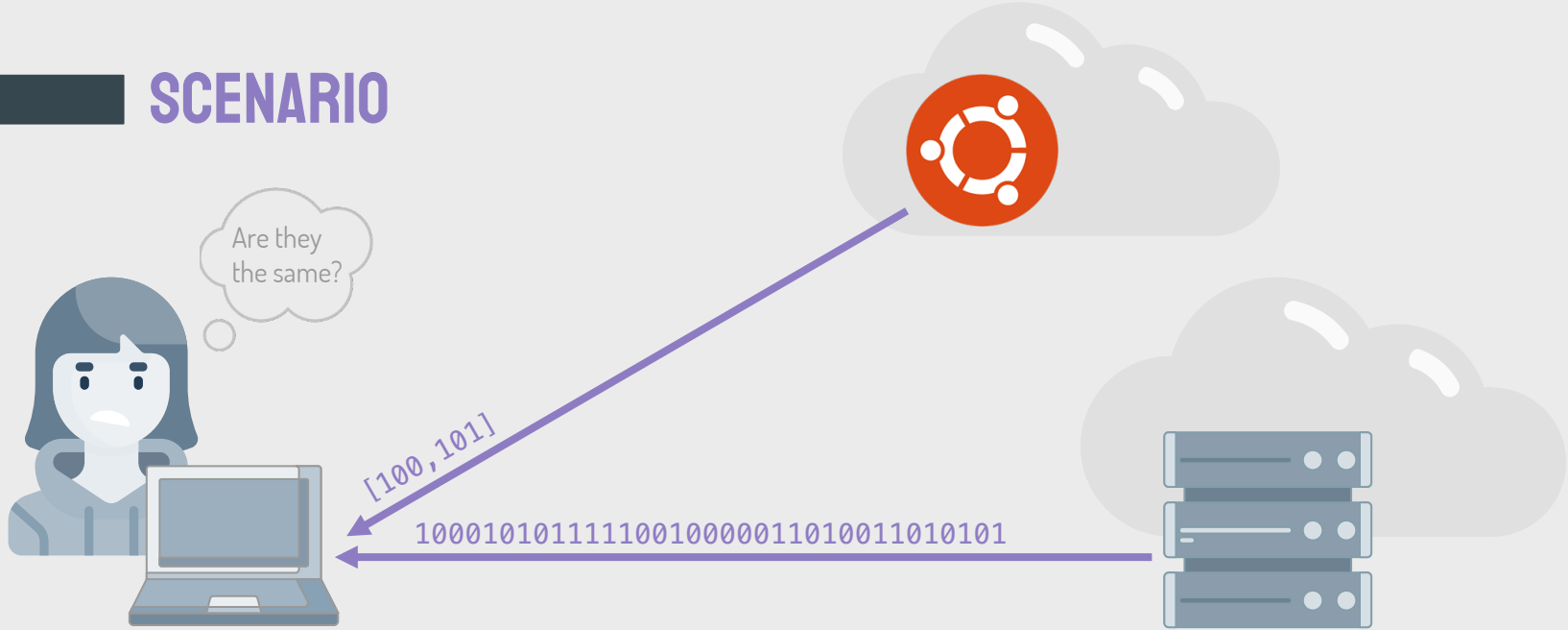
SCENARIO



You could *also* download the official file from Ubuntu and compare them

But that's double the work, double the download, and defeats the point of the cloud infrastructure

SCENARIO

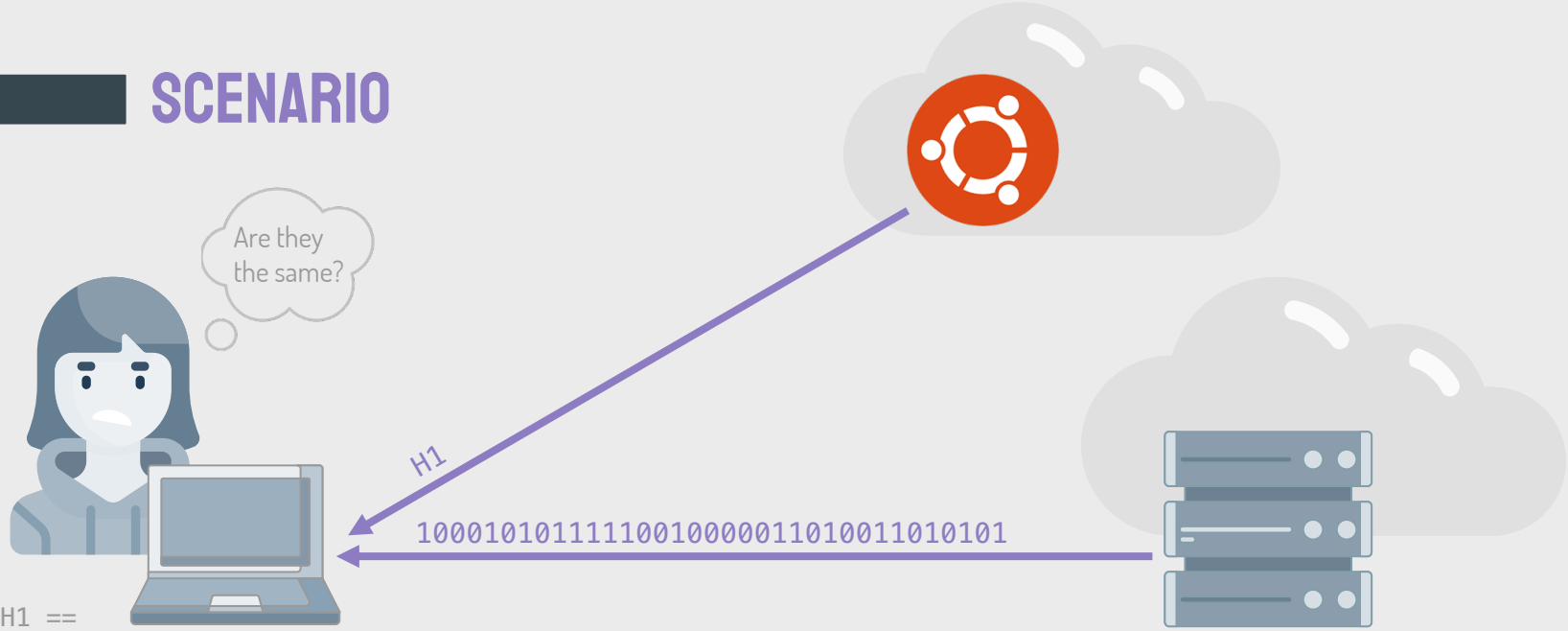



```
s=100010101111100100
00011010011010101
t=[100,101]
Assert s[:3]==t[0]
Assert s[-3:]==t[1]
```

You could check the first and last few bits

But the malware could have been inserted in between

SCENARIO




```
Assert H1 ==
Checksum(100
010101111100
100000110100
   11010101)
```

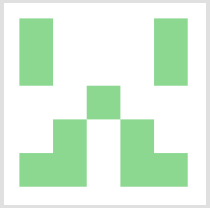
You could have Ubuntu send you a short error-detecting code

This is designed to detect random errors. It won't stop bad guys from constructing false-negatives.

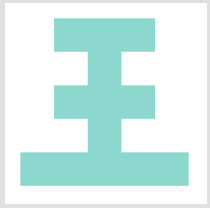
DATA FINGERPRINTS?

What if there was a way to assign a short, unique and easy-to-generate fingerprint to any string?

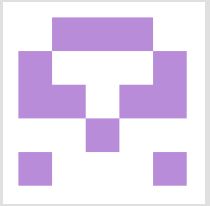
"Anyone who attempts to generate random numbers by deterministic means is, of course, living in a state of sin."



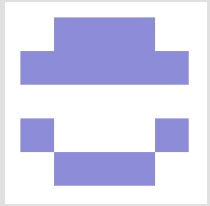
"The enemy knows the system."



"If you reveal your secrets to the wind, you should not blame the wind for revealing them to the trees."

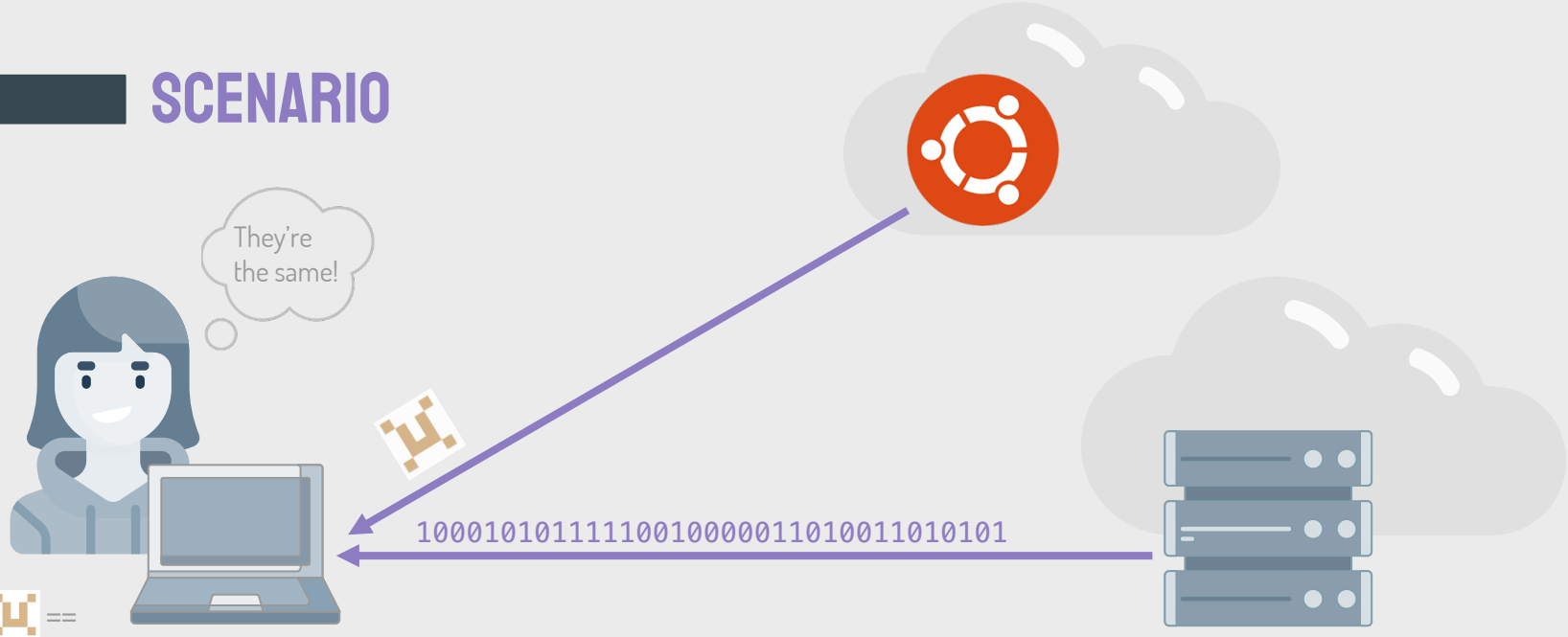


"The trouble with quotes on the Internet is that you never know if they are genuine."



http://identicon.net/

SCENARIO



You could have Ubuntu send you the fingerprint

The fingerprint is unique... or at least really really hard to find another string that generates the same fingerprint

CRYPTOGRAPHIC HASHES ALL OVER

DATA FINGERPRINTING

Identifying malware, genuine code, git intrusion detection, chain-of-custody

DIGITAL SIGNATURES

Efficiency aid for public-key cryptography

MESSAGE AUTHENTICATION CODES

Guaranteeing message integrity

KEY-DERIVATION

Turn one secret key into a bunch of keys

SECURE PASSWORD STORAGE

Store hashes instead of passwords to mitigate breaches

PROOF OF WORK

Primary consensus mechanism of Bitcoin and other cryptocurrencies

NON-INTERACTIVE ZKPs

A building block in non-interactive zero-knowledge proofs (Fiat-Shamir heuristic)

POST-QUANTUM CRYPTO

Building block in several post-quantum algorithms (key agreement, signatures

SECURE PROTOCOLS

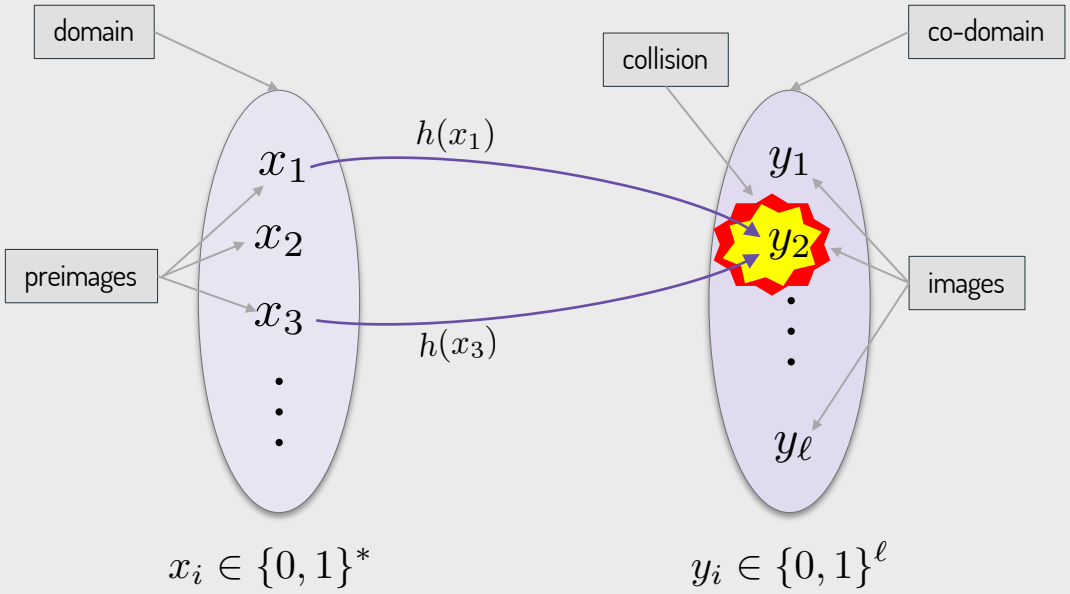
A building block of secure protocols e.g. commitments, exact matching, time-stamping

WHAT IS A HASH FUNCTION?

$$h : \{0,1\}^* \rightarrow \{0,1\}^\ell$$

A hash function maps arbitrary-length strings to fixed-length (l-bit) strings

TERMINOLOGY



WHAT IS A HASH FUNCTION?

Example: Let h(x) = x mod 256

$h : \{0,1\}^* \rightarrow \{0,1\}^8$

Pre-image, x	Image, h(x)
…	…
41685114102567	39
41685114102568	40
41685114102569	41
41685114102570	42
…	…

Problem: Related images (hashes) have related pre-images

WHAT IS A *CRYPTOGRAPHIC* HASH FUNCTION?

$$h : \{0,1\}^* \rightarrow \{0,1\}^\ell$$

A cryptographic hash function is a pseudo-random hash function

WHAT IS A *CRYPTOGRAPHIC* HASH FUNCTION?

Example: Let h(x) = “random oracle”

$h : \{0,1\}^* \rightarrow \{0,1\}^8$

Pre-image, x	Image, h(x)
…	…
41685114102567	143
41685114102568	35
41685114102569	69
41685114102569	193
…	…

Each 8-bit image was chosen by an independent 8-coin coin toss

Related pre-images have maximally unrelated images

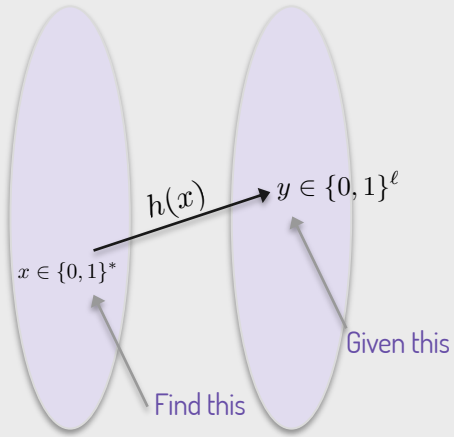
RANDOM ORACLES DON'T ACTUALLY EXIST

- **Impossible**: If there are infinitely many possible inputs, a random oracle requires infinite memory to maintain image/preimage pairs
- **Impractical**: You could bound the input, but it's still exponential memory in the hash bit length
- **Fake it till you make it**: In practice we only simulate random oracles using (hopefully!) highly non-linear functions that can't be easily inverted

SECURITY PROPERTIES

- **Pre-image resistance**: Given a hash, it should be hard to find a message that produces that hash
- **Second preimage resistance**: Given a message, it should be hard to find *another* message that produces that hash
- **Collision resistance**: It should be hard to find any pair of messages that collide (hash to the same value).

PRE-IMAGE RESISTANCE

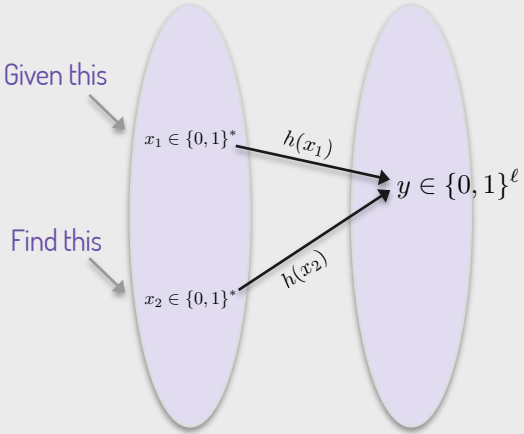


What is the probability of finding a preimage if h( ) is a random oracle?

- Throw a ball into a random bin
- P(bin i not empty) = 1-P(bin i empty)
- P(bin i empty after 1 throw) = $\frac{2^\ell - 1}{2^\ell}$
- P(bin i empty after k throws) = $\left(\frac{2^\ell - 1}{2^\ell}\right)^k \approx \frac{k}{2^\ell}$
- P(bin i not empty after $2^{\ell-1}$ throws) = $\frac{2^{\ell-1}}{2^\ell} = \frac{1}{2}$

An ideal ℓ-bit hash function provides ℓ-bits of preimage resistance

SECOND PRE-IMAGE RESISTANCE



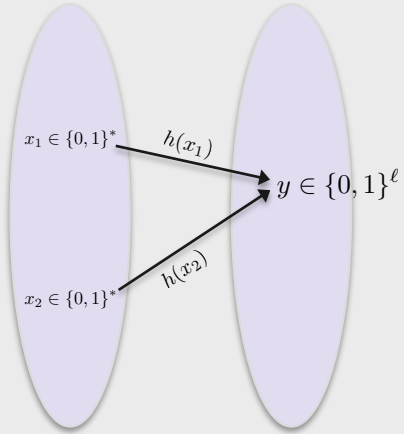
What is the probability of finding a second preimage if h( ) is a random oracle?

- Similar analysis to pre-image resistance

An ideal ℓ-bit hash function provides ℓ-bits of second preimage resistance

COLLISION RESISTANCE

Find any pair such that



What is the probability of finding a collision if h( ) is a random oracle?

- Analysis based on the birthday paradox: how many people do you need in the room to expect some pair share a birthday? Answer: only 23.
- Intuition: if there are k days, and O(sqrt(k)) people in a room, there are O(sqrt(k)^2)=O(k) pairs of people.
- Each pair of birthdays differs by 0..(k-1)/2 days, so after seeing O(k) pairs, you would expect to see a pair the differed by 0 days.

An ideal ℓ-bit hash function provides $\frac{\ell}{2}$-bits of collision resistance

BITS OF SECURITY

If you require 128-bits of pre-image resistance, which of these hash functions are acceptable?

If you require 128-bits of collision resistance, which of these hash functions are acceptable?

Hash algorithm	Output bit length
MD5	128
SHA-1	160
SHA-256	256
SHA-512	512

HASHING: TRY IT YOURSELF IN PYTHON

Hashing different messages with the same hash function:

```
>>> import hashlib
>>> hashlib.sha224(b"Hello1").hexdigest()
'cddd99351fcf09db06222975af5a8c6a5f01f373e37062eeed65db99'
>>> hashlib.sha224(b"Hello2").hexdigest()
'4b35f88bf7e4396e6fb41fed2305592beeda45b4e782f7c0d998db6f'
>>> hashlib.sha224(b"Hello3").hexdigest()
'1122550736f8baea19830d4f6f53fced3b9f28f691728823d62d4642’
```

HASHING: TRY IT YOURSELF IN PYTHON3

Hashing the same message with different hash functions:

```
>>> hashlib.md5(b"Hey").hexdigest()
'd0eedb799584d850fdd802fd3c27ae34'
>>> hashlib.sha1(b"Hey").hexdigest()
'e4599fa9f2653074005dad27f086837c20faeef4'
>>> hashlib.sha256(b"Hey").hexdigest()
'581d43745726e0ee62911178bfb3887c3fe295d29eeb741f0e40f91e8a70907a'
>>> hashlib.sha512(b"Hey").hexdigest()
'ec90c352aac8deb3e15d399f719ee3aa0a9e2dcb4d197cfb32c0314e216c5e861
6f3193791421150967ee0ef97cfcebae1928612222800eea1bc3fb45598736d'
>>> hashlib.blake2b(b"Hey").hexdigest()
'dfaad2d4f5391cfab4440d692d45aea4d81c083d194f5d84e4d193aecd85f2b6a
81f969ce012080de78b7329a6e5718c1846e17e8a9647f1e8e574f543426f18'
```

COLLISIONS IN MD5

Uh oh

```
>>> import hashlib
>>>
hashlib.md5(bytes.fromhex("4dc968ff0ee35c209572d4777b72158
7d36fa7b21bdc56b74a3dc0783e7b9518afbfa202a8284bf36e8e4b55b
35f427593d849676da0d1d55d8360fb5f07fea2")).hexdigest()
'008ee33a9d58b51cfeb425b0959121c9'
>>>
hashlib.md5(bytes.fromhex("4dc968ff0ee35c209572d4777b72158
7d36fa7b21bdc56b74a3dc0783e7b9518afbfa200a8284bf36e8e4b55b
35f427593d849676da0d155d8360fb5f07fea2")).hexdigest()
'008ee33a9d58b51cfeb425b0959121c9'
```

MD5 IS NO LONGER COLLISION RESISTANT

COLLISIONS CAN BE FOUND FASTER THAN BRUTE-FORCE SEARCH!!

FINAL POINTS ABOUT HASHING

- Hashing is deterministic. Hash the same message twice, get the same output
- Don't get confused: Hashing is not encryption! It doesn't have a decryption function or a key
- Every hash function has collisions (there are more balls than bins)

QUESTIONS?

Contact Prof. Essex:
aessex@uwo.ca
@aleksessex

See course website for slides and videos:
https://whisperlab.org/security