EECS 388: Lab 2

Length extension
Hash collisions

Upcoming Deadlines

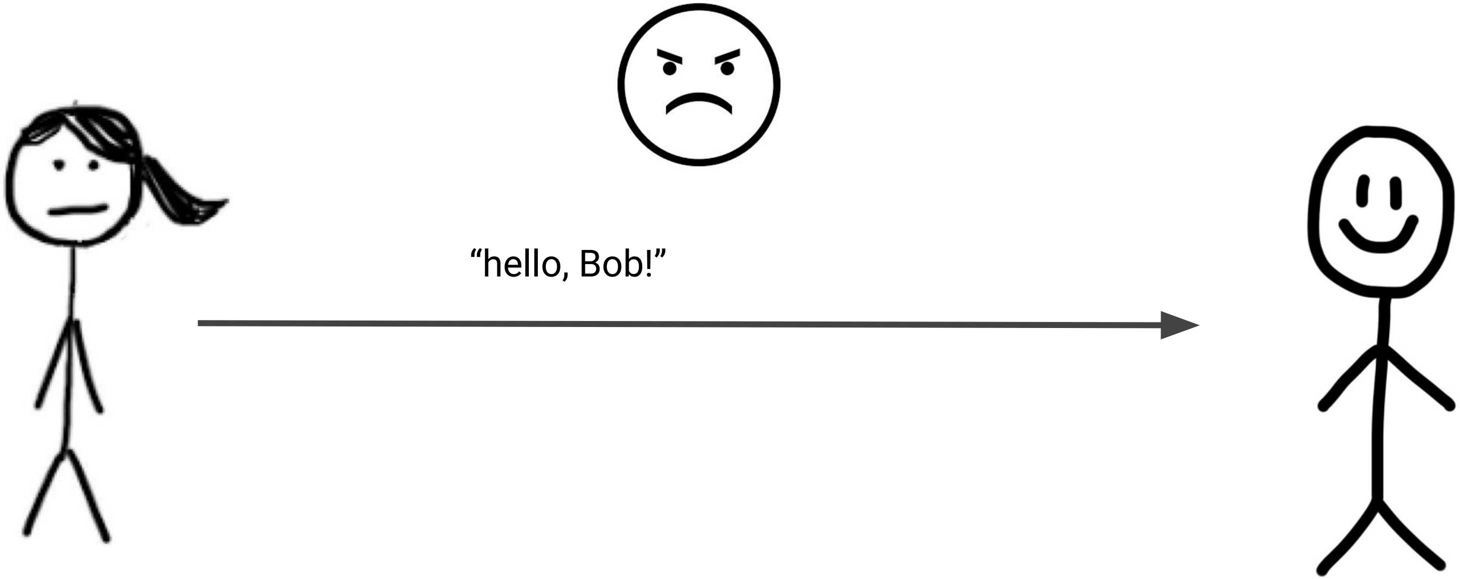
- Project 1: Cryptography
  - Part 1 due: **Thursday, September 12 at 6 p.m.**
    - Coverage: Length extension attack, Hash collisions
  - Part 2 due: Thursday, September 19 at 6 p.m.
    - Coverage: Padding oracle attack, RSA signature forgery

Reminder: Canvas quizzes due the day before the next lecture

Length Extension Attack

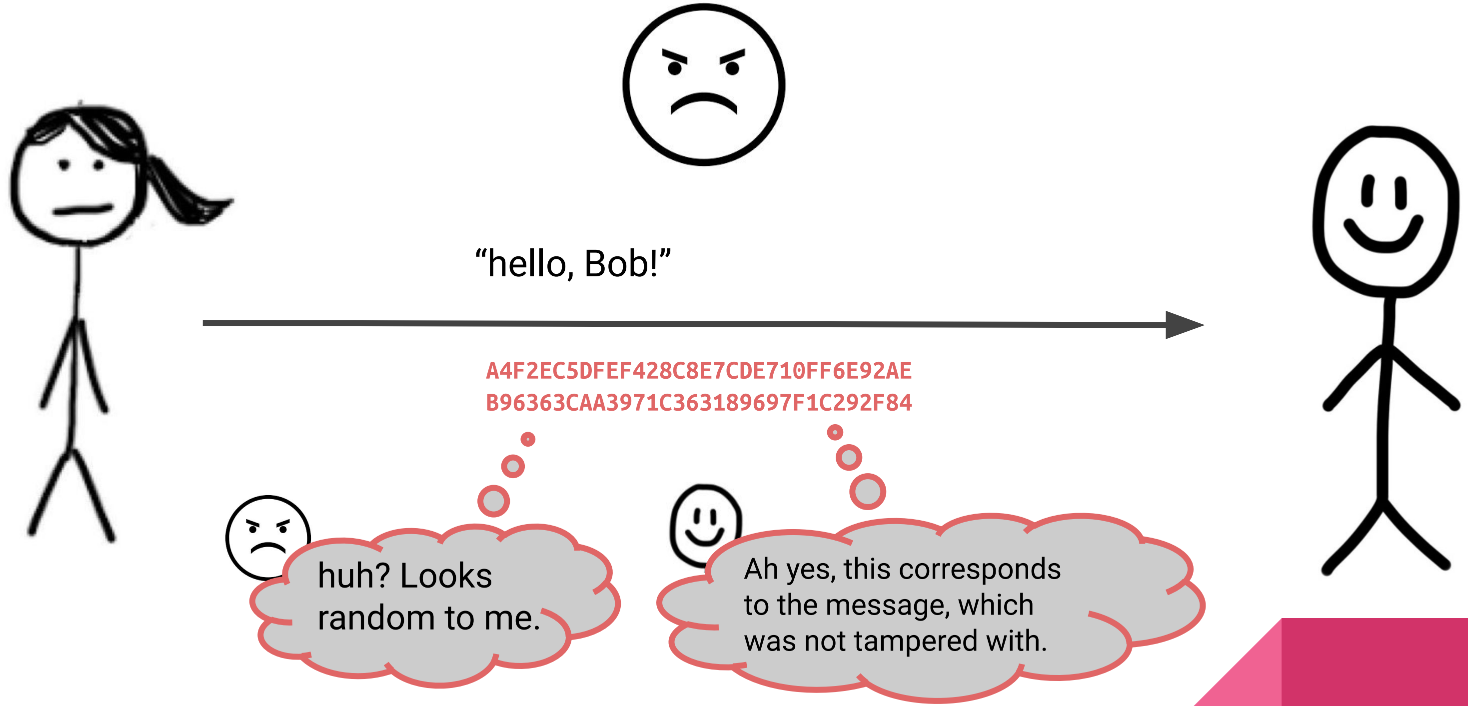
(Project 1, Part 1.1)

Alice messages Bob



What could possibly go wrong?

Alice messages Bob, with a MAC



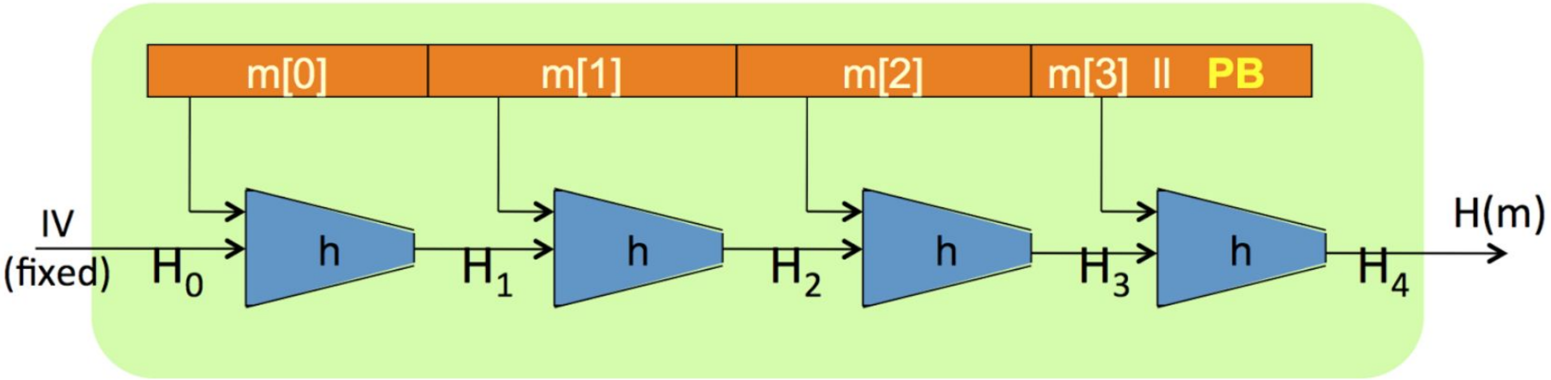
What is this `A4F2EC5DFEF428C8E7CDE710FF6E92AE B96363CAA3971C363189697F1C292F84` …?

Output from a pseudorandom function; let's call it $f_k(x)$.

- In this case, x = "hello, Bob!"
- f_k is indistinguishable from a random function, unless you know k.
  - Alice and Bob know k, so they know **$f_k(x)$ for all x**.
  - Mallory doesn't know k, and cannot derive **$f_k(x)$**.
- Embodied by functions like HMAC-SHA256
  - Does not mean all MACs are PRFs
- What happens if we use plain SHA-256 instead?

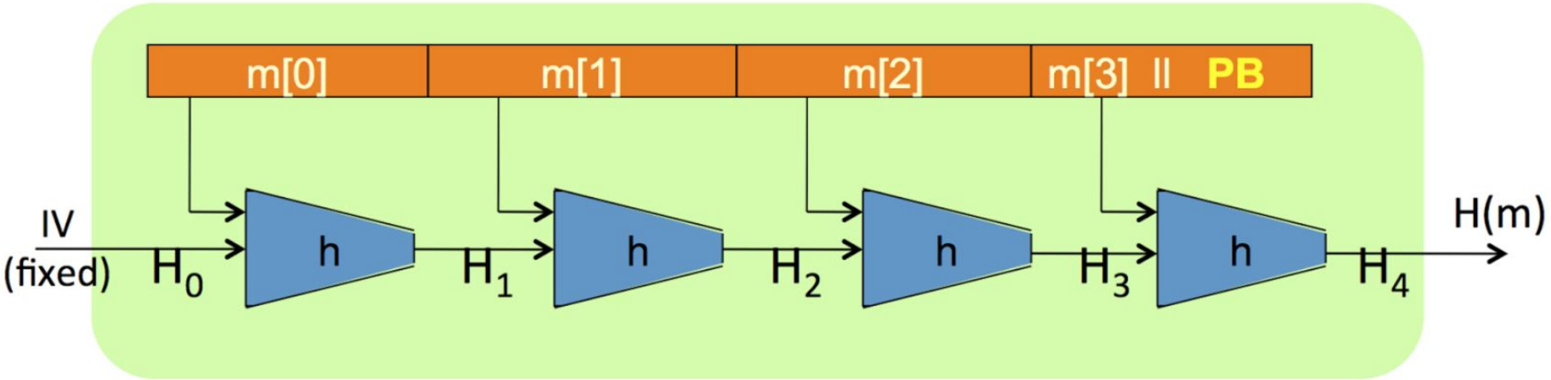
Common Hash Function Construction

- MD5, SHA-1, SHA-256: Hash functions utilizing **Merkle-Damgård Construction**



- **Merkle-Damgård Construction:** uses a compression function (h) of small, fixed-size inputs to construct a bigger hash function (H) of variable-length input

Merkle-Damgård Construction

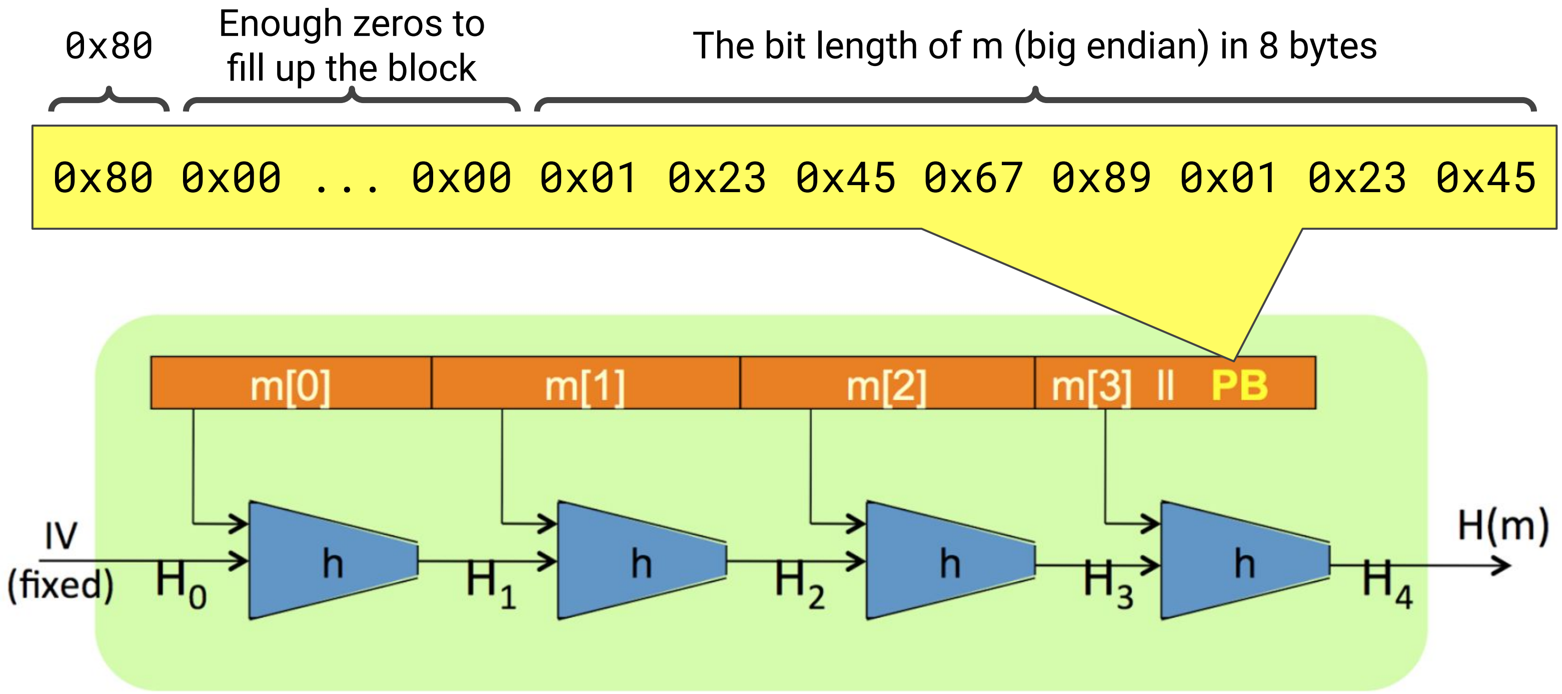


Given $\mathbf{h: T \times X \longrightarrow T}$ (compression function)

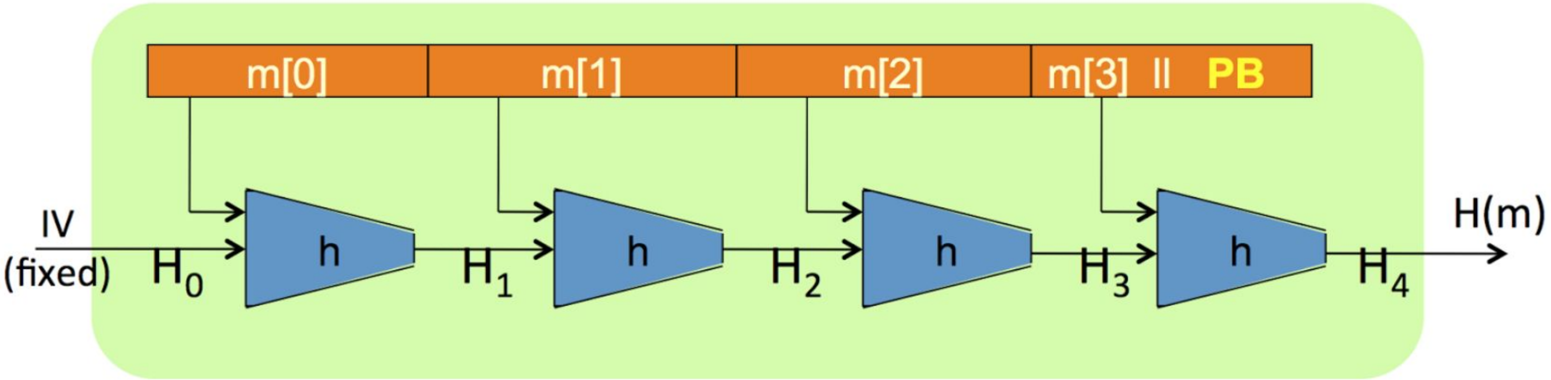
we obtain $\mathbf{H: X^{\leq L} \longrightarrow T}$. H_i - chaining variables

PB: padding block

Merkle-Damgård Construction



Merkle-Damgård Construction

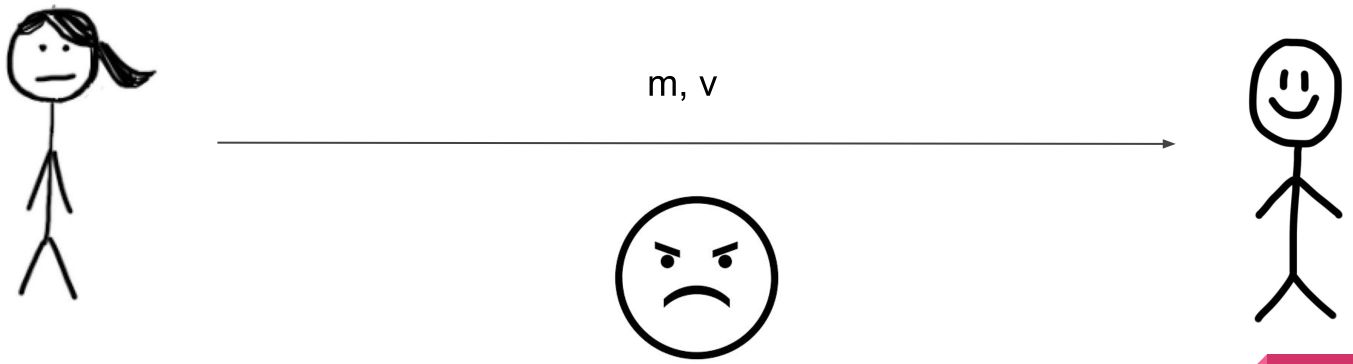


Key (proven!) property: if **h** is collision-resistant, then **H** is also collision-resistant.

Makes building a secure hash function easier!
But… introduces length-extension vulnerability :(

Length Extension Attack

- Alice and Bob want to communicate and have integrity
- They have a shared secret key, k, and hash function, H



Verifies that
H(k || m) == v

m = “hello, Bob!”
v = H(k || “hello, Bob!”)

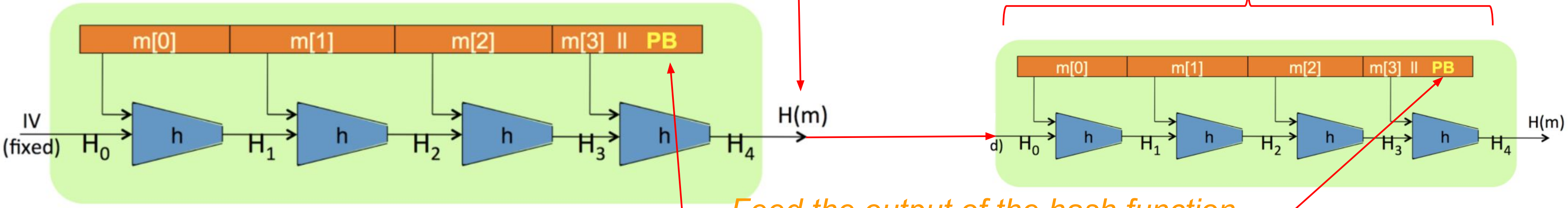
Mallory intercepts m and v
… what can she do?

Length Extension Attack

She doesn't know this message because it had a secret k, but she doesn't need it….

This is the v that Mallory intercepted

Appended message can be whatever Mallory wants, like "please send $1000"



Feed the output of the hash function into the next SHA-256 block

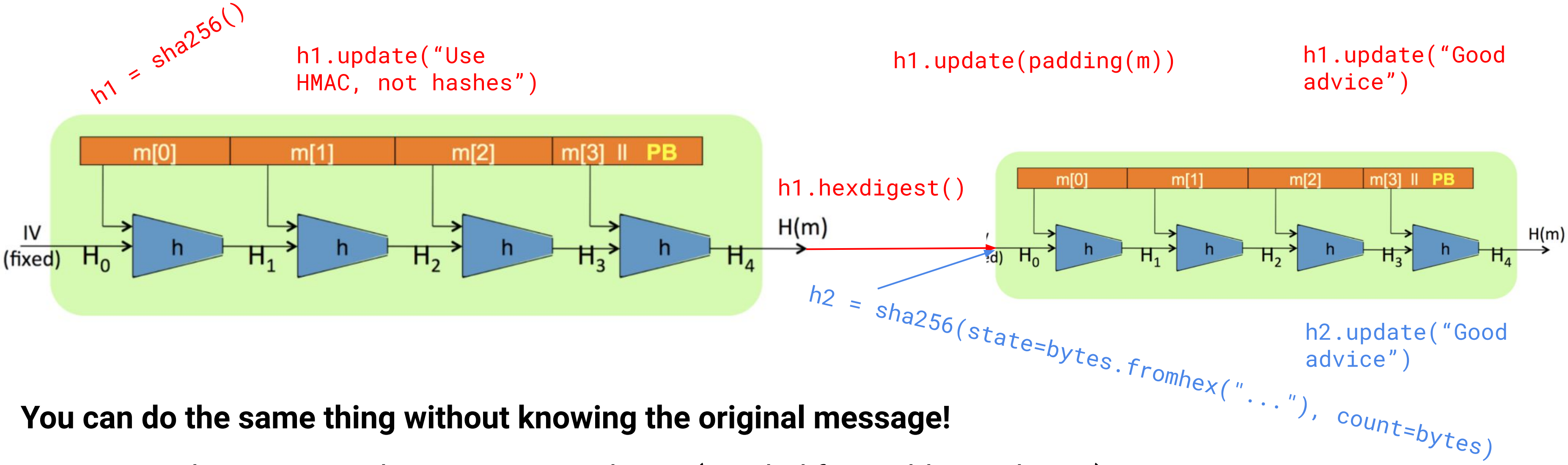
- Now Mallory can send m = original_message || padding || "please send $1000" ***with a valid v that Bob will accept***

Just need to know the length of the original message so the padding block has the correct length

Length Extension Attack

How can we “extend” a hash computation?

Thought experiment (`h1`): ***Imagine you know the original message…***



You can do the same thing without knowing the original message!

`h2`: Start with **`hash(m)`** and ***`length of m`*** in bytes (needed for padding scheme)

We didn’t need to know “Use HMAC, not hashes”, only its hash!

What have we hashed?

h1:

Use HMAC, not hashes\x80\x00\x00\x00\x00…\x00\xa0Good advice

h2:

Use HMAC, not hashes\x80\x00\x00\x00\x00…\x00\xa0Good advice

Important takeaways

- Both hash the same thing
- Both hash padding in the middle

Length Extension Attack: Prevention

- Length extension attack comes from using plain hash functions as a MAC
- It doesn't mean that the hash function itself is a bad hash function
- To make a good MAC function from a hash function, use the HMAC construction, as discussed in lecture:

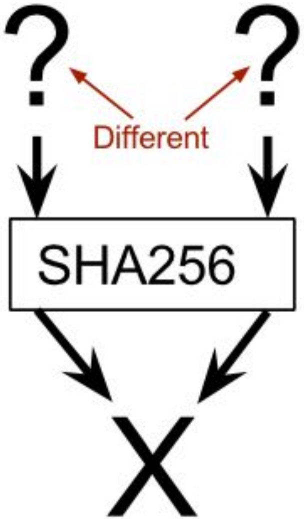
$$\text{HMAC}_k(m) = \text{hash}(k \oplus c_1 \,\|\, \text{hash}(k \oplus c_2 \,\|\, m))$$

Spec Demo

```
from pysha256 import sha256, padding

m = 'Use HMAC, not hashes'.encode() # .encode() converts str to bytes

h1 = sha256()

h1.update(m)

print(h1.hexdigest()) # '8f36ee4a3885bcc8a8446b09e9498808c97667f0266e3641c5d2abca457f9187'

padded_message_len = len(m) + len(padding(len(m)))

h2 = sha256(state=bytes.fromhex('8f36ee4a3885bcc8a8446b09e9498808c97667f0266e3641c5d2abca457f9187'),
count=padded_message_len)

x = 'Good advice'.encode() # .encode() converts str to bytes

h2.update(x)

print(h2.hexdigest()) # verify that it equals the SHA-256 hash of m + padding(len(m)) + x
```

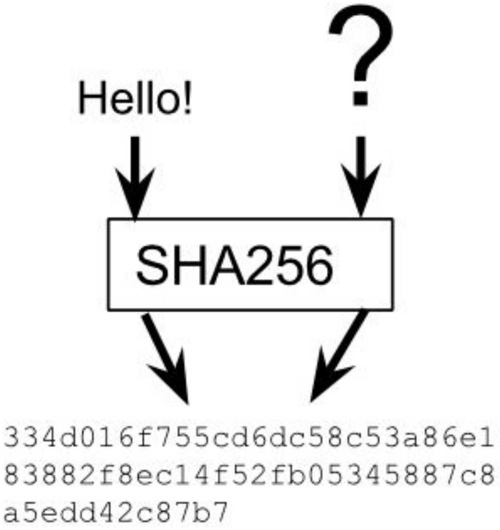
More with
Hash Collisions

Properties of *Good Cryptographic* Hash Functions



Collision resistance

Preimage resistance



Second-preimage resistance

Properties of *Good Cryptographic* Hash Functions

And how to break them

- Preimage resistance
    - For a given output, find an input that produces it.
- Collision resistance
    - Find *two inputs* that map to the same output.
- Second-preimage resistance
    - For a *given* input, find a *different* input with the same output.

Collision resistance implies *second-preimage* resistance, but not *preimage* resistance.

Second preimage *attack* implies collision *attack*.

RIP MD5 and SHA-1

- ❌ Preimage resistance
  - Theoretically broken
- ❌ Collision resistance
  - Totally broken
- ❌ Second preimage resistance
  - Theoretically broken, based on same attack above

RIP MD5 and SHA-1

Collisions with an identical prefix

MD5: 2004 (cost in 2023: free, you do this using *fastcoll*!)

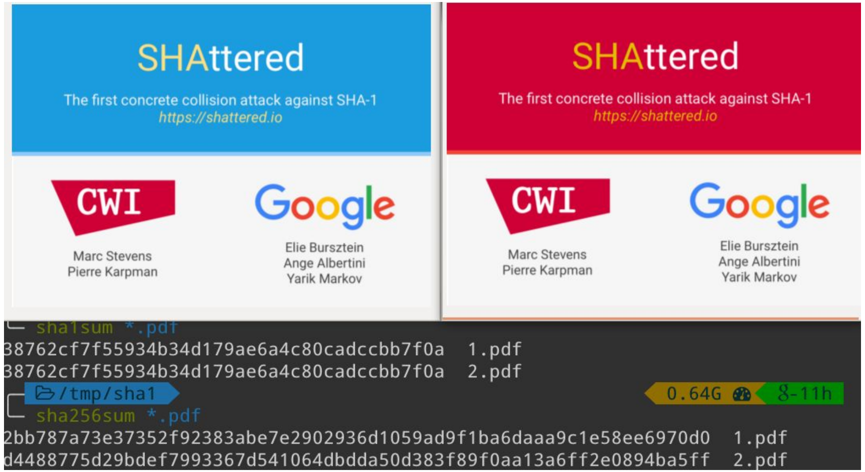
SHA-1: 2017 (cost in 2023: ~$10,000)

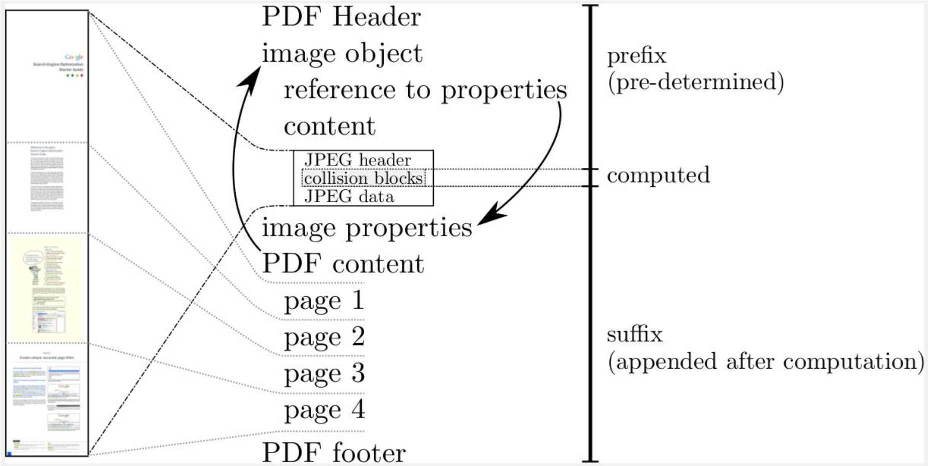
Collisions with a (chosen) pair of different prefixes

MD5: 2007 (cost in 2023: ~$5)

SHA-1: 2019 (cost in 2023: ~$75,000)

SHA-1 Collisions!





https://shattered.io/

Review: Hashing vs Encryption?

When do we use a **hash function** vs. a **MAC** vs. a **cipher**?

Hash function	MAC	Cipher

Review: Hashing vs Encryption?

When do we use a **hash function** vs. a **MAC** vs. a **cipher**?

Hash function

- Check if 2 files are the same

MAC

- Message integrity

Cipher

- Message confidentiality

See you next week!