

# Day 3: Hash Functions

---

and MACs

↑


Message  
Authentication  
Codes

---

---

---

---



Today:

① Hash Functions

② Padding

③ Daisy Chain.

④ Interlude: Formal Security

② MACs

# 1 Hash Functions.

"Definition" A secure hash function

$$f: X \rightarrow Y$$

↑

↖

↙

function input space output space

(deterministic large)

(smaller: ~256 bits)

is a function that "looks random".

Q: How do hash functions differ from PRGs which also 'look random'?

PRGs: small input (~256b), big output (n bits)

Hashes: big input (n bits), small output (~256b)

↳ they're opposites.

As 'random functions', hash functions have many desirable properties:

→ Hard to find  $x_1 \neq x_2$  s.t.  $H(x_1) = H(x_2)$

↪ For given  $y$ , hard to find  $x$  s.t.  $H(x) = y$

"Collision Resistant" "Pre-image Resistant" ↗

[1a] Q: How do we build good hash fns?  
↳ too hard for this class (dark magic)

Today: How do we build a big hash fn from a small hash fn?

big: input: n-bytes      output: 32 bytes

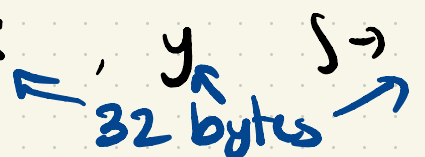
small: input: 64-bytes      output: 32 bytes

Construction

The "daisy-chain hash" (a.k.a. Merkle-Damgård hash).

Given a "compression" hash

$C(x, y) \rightarrow z$ ,  
Define



$H(d)$  n bytes

in two stages.

Stage 1: Padding

1. Add 0's to the end of  $d$  until its length (in bytes) is divisible by 32.

examples

zero byte

$$b'ab' \rightarrow b'ab' + \overbrace{b'\backslash x00'}^{30} * 30$$

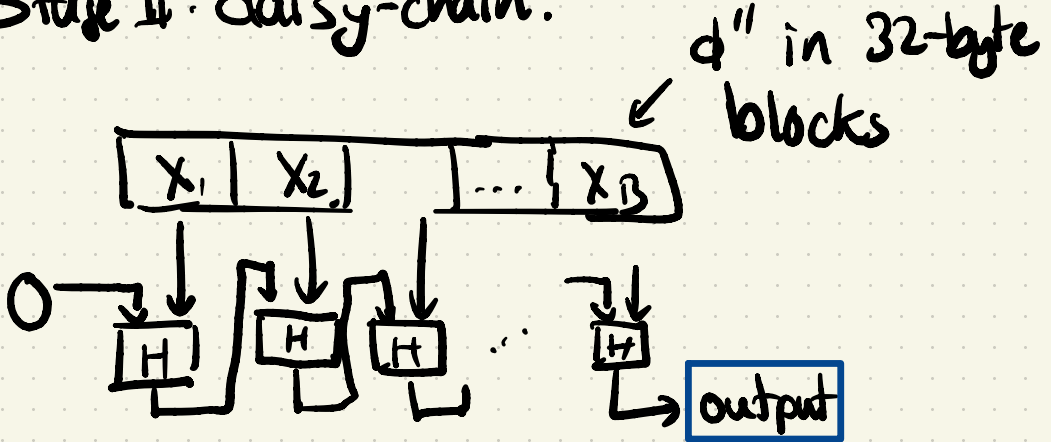
$$b'a'^{32} \rightarrow b'a'^{32} \text{ (no change!)}$$

call the result  $d'$

2. Add the length of  $d$  as a 32 byte block to the end of  $d'$

$$d'' = d' + \text{bitstring\_to\_bytes}(f'\{\text{len}(d):0256b\}')$$

Stage II: daisy-chain:



↑  
final output.  
the "hash value"

daisy-chain in pseudocode:

```
fn daisyChain (d'')
```

```
  acc = 32 0 bytes. (b'\x00' * 32)
```

```
  for 32-byte block b in d'':
```

```
    acc = Compress (acc, b)
```

```
  output acc.
```

Q: Why do we add 0-bytes during padding?

→ So the data splits evenly into 32-byte blocks.

Q: Why do we add the length?

→ If we didn't, then

$$H(x \parallel b) = \text{Compress}(H(x), b)$$

↗ structured relationship between

$H(x||b)$  and  $H(x)$

→ A "random" hash should have any structured relationships!

1c Interlude: Formal Security Definitions for Hash Functions

Recall Collision Resistance (Informal): "Hard to find  $x \neq x'$  such that  $H(x) = H(x')$ "

Definition: A pair  $(x, x')$  is a collision for  $H$  if  $x \neq x'$  and  $H(x) = H(x')$

It's supposed to be "hard" to find a collision.

What does "hard" mean?

- It should take a long time?
- the chance of doing it should be small?

Both!

Defn (formal): A hash function  $H_\lambda$  is collision-resistant if for all efficient algorithms  $A$ ,

$\Pr \left[ \begin{array}{l} x \neq x' \text{ and } H_\lambda(x) = H_\lambda(x') \\ \text{when } (x, x') \leftarrow A(\lambda) \end{array} \right]$  is negligible in  $\lambda$

$H_\lambda$ : a hash function parameterized on the output size:  $\lambda$ . Idea: bigger  $\lambda \rightarrow$  harder to find a collision.

"efficient algorithm": a (possibly randomized) program whose runtime is  $\leq$  some polynomial  $p(\lambda)$ .

- eg.
- Python function
  - Turing Machine.

"negligible in  $\lambda$ ": asymptotically smaller

than  $\frac{1}{\lambda^c}$  for all  $c \in \{1, 2, \dots\}$

that is,  $f(\lambda)$  is negligible in  $\lambda$  if

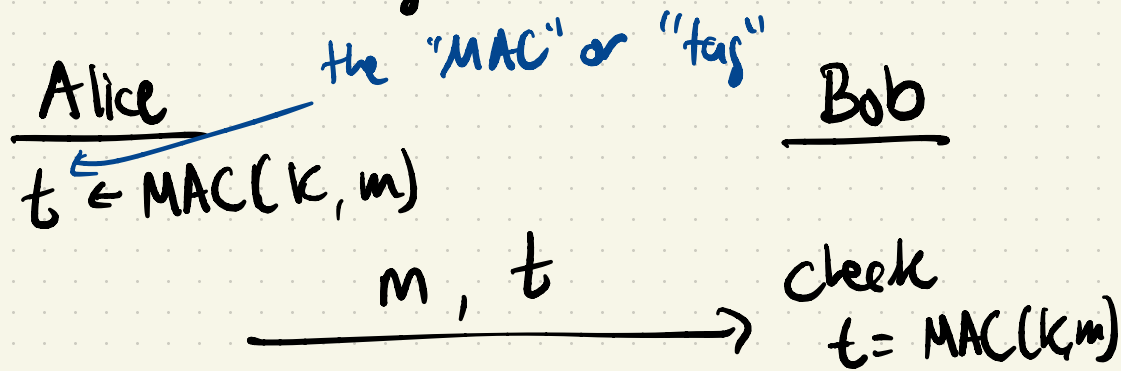
$$\forall c \in \{1, 2, 3, \dots\}, \quad \lim_{\lambda \rightarrow \infty} \frac{f(\lambda)}{\lambda^c} = 0.$$

example:  $\frac{1}{2^\lambda}$  is negligible in  $\lambda$ .



## 2 Message Authentication Codes:

Goal: Detect if the messenger modifies a message.



Q: Is  $\text{MAC}(k, m) = H(m)$  a secure MAC?

No! Messenger can trick Bob by changing too!

$$m' \neq m, t' \leftarrow H(m')$$

We have to ensure that the messenger

doesn't know how to change  $t...$

Hash-MAC:

+ in Python

$$\text{MAC}(m, k) = H(k \parallel m)$$

↑  
ensures that messenger  
can't evaluate MAC...

Combining a cipher  $E = (\text{Enc}, \text{Dec})$   
and a MAC:

Alice

$$ct \leftarrow \text{Enc}(m, k)$$

$$t \leftarrow \text{MAC}(ct, k')$$

use  
two  
keys

Bob

$ct, t$

$$t \stackrel{?}{=} \text{MAC}(ct, k')$$

$$m \leftarrow \text{Dec}(ct, k)$$

- Gives "Authenticated Encryption"

- This approach is called "Encrypt then MAC"  
"ETM"

For the problems:

32 bytes

cryptoy. compress (a, b) → c.

Problems:

- Implement padding
- Implement digital signature hashing
- Implement hash-based MAC!

