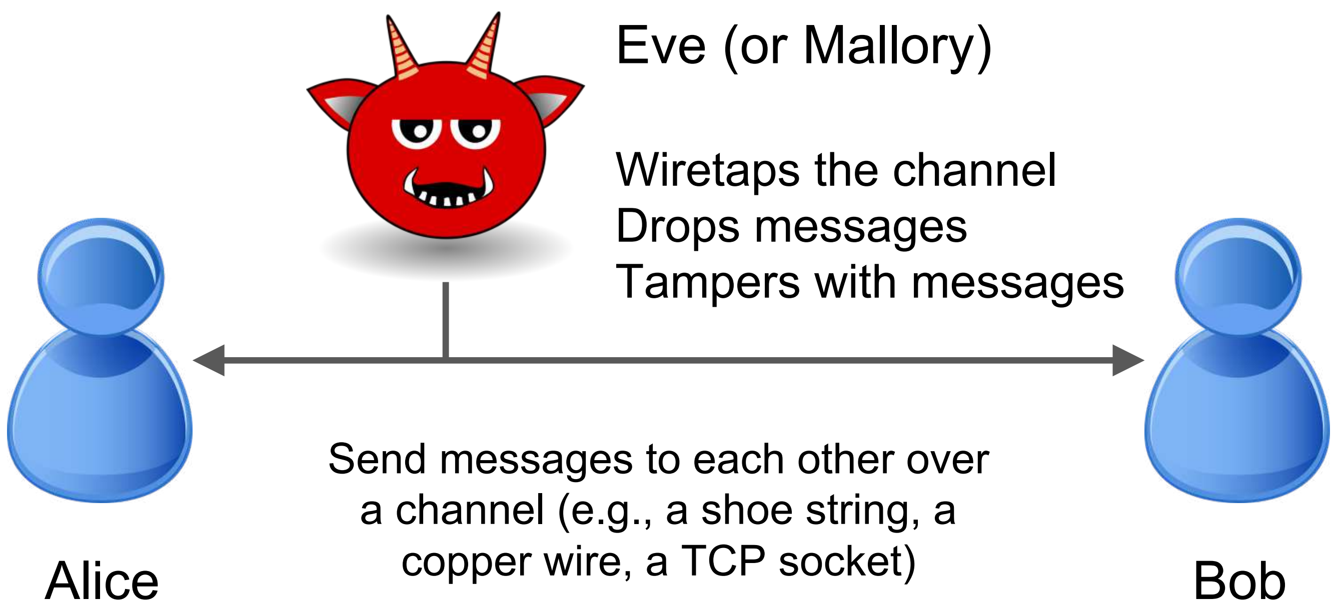
Lecture 18 – Message Integrity

Stephen Checkoway
University of Illinois at Chicago
CS 487 – Fall 2017
Slides from Miller & Bailey's ECE 422

Cryptography is the study/practice of techniques for **secure communication**, even in the presence of powerful **adversaries** who have **control** over the **underlying channel**



Learning goals of cryptography module

- Understand the interfaces of basic crypto primitives

Hashes, MACs, symmetric encryption, public key encryption, digital signatures, key exchange

- Apply the adversarial mindset to crypto protocols

- Appreciate the following warning:

"Don't roll your own Crypto!" …….

- Familiarity with concepts, vocabulary

Lectures are for breadth

Cryptography is not just encryption!

Cryptography can help ensure:

- Confidentiality: secrecy, privacy
- Integrity: tamper resilience
- Availability
- Non-repudiability, or deniability

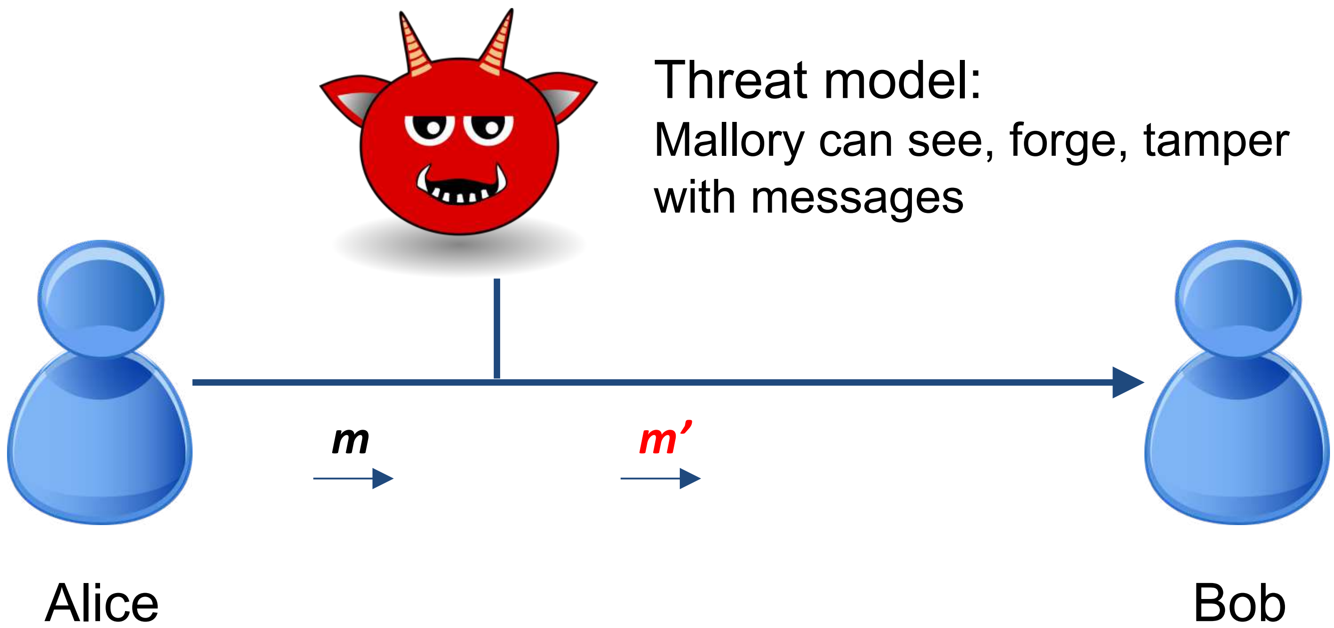
…. many more properties

Message Integrity

Hashes, MACs

Goal: Secure File Transfer

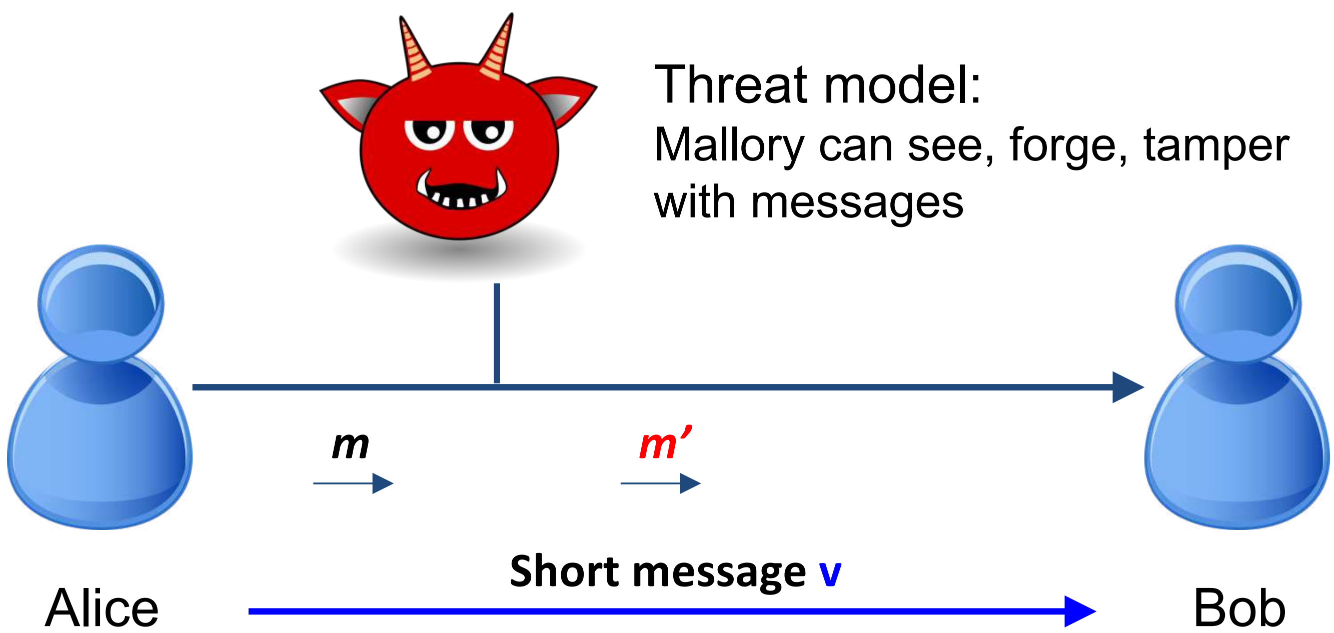
Alice wants to send file ***m*** to Bob (let's say, a 4 Gigabyte movie)
Mallory wants to trick Bob into accepting a file Alice didn't send



Threat model:
Mallory can see, forge, tamper with messages

Goal: Secure File Transfer

Alice wants to send file ***m*** to Bob (let's say, a 4 Gigabyte movie)
Mallory wants to trick Bob into accepting a file Alice didn't send

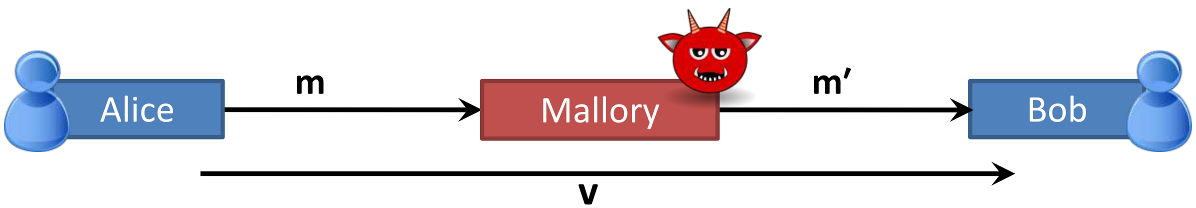


Setup assumption: *Securely transfer a short message!*

Solution: Collision Resistant Hash Function (CRHF)

Hash Function ***h***: $\{0,1\}^* \to \{0,1\}^{256}$ (or other fixed number)

1. Alice computes **v** := ***h***(**m**)
2. Alice transfers **v** over secure channel, **m** over insecure channel



3. Bob verifies that **v** = ***h***(**m'**), accepts file iff this is true

Function ***h*** ? We're sunk if Mallory can compute **m' ≠ m** where ***h***(**m**) = ***h***(**m'**)! A *collision*!

Contrast with: "checksums" e.g. CRC32.... defend against random errors, not a deliberate attacker!

Hash function properties

Good hash functions should have the following properties

Which of these properties implies which others?

First pre-image resistance:

Given h(m), it is computationally infeasible to find m' s.t. h(m') = h(m)

Second pre-image resistance:

Given m_1, it is computationally infeasible to find $m_2 \neq m_1$ s.t. $h(m_1) = h(m_2)$

Collision resistance:

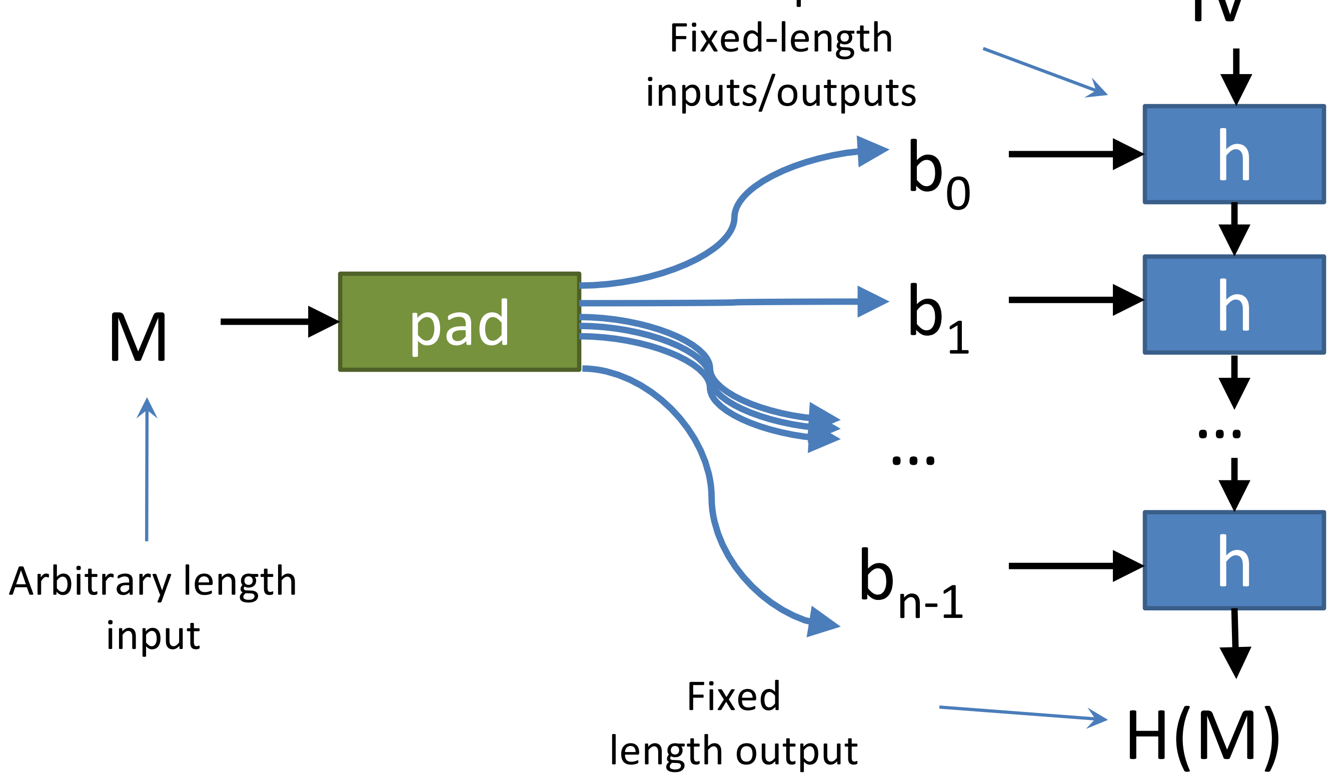
It is computationally infeasible to find *any* $m_1 \neq m_2$ s.t. $h(m_1) = h(m_2)$

Hash function construction

- Merkle–Damgård construction
  - Pad message to a multiple of block size
  - Run a compression function over each block and the output of the previous compressed block (see next slide)
  - Used for MD5, SHA-1, SHA-2
- Sponge construction
  - Pad message to a multiple of a fixed size (the bitrate r)
  - “Absorb” the message r bits at a time by XORing with part of the internal state, and permuting the whole state by permutation f
  - “Squeeze” out the output r bits at a time, applying f in between
  - SHA-3

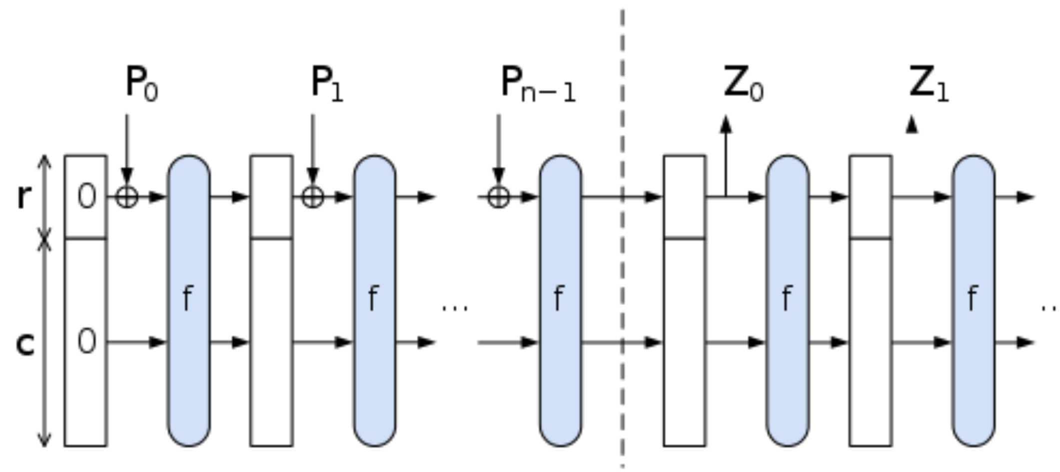
Merkle–Damgård Construction

- Arbitrary-length input
- Fixed-length output
- Built from fixed-size “compression function”



Sponge construction

- Internal state initially 0
  r+c total bits
- P_i are message blocks
- Z_i are the output blocks



What is SHA256?

`$ sha256sum file.dat`

Cryptographic hash

Input: arbitrary length data
(No key)
Output: 256 bits

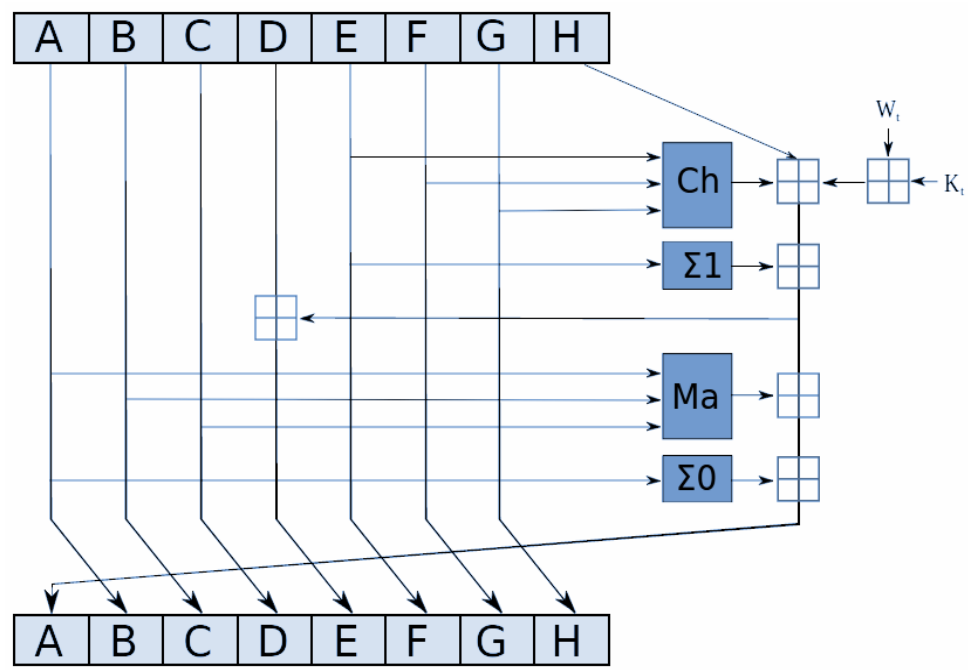
Built with compression function, ***h***

(256 bits, 512 bits) in → 256 bits out

Designed to be really hairy (64 rounds of this)!

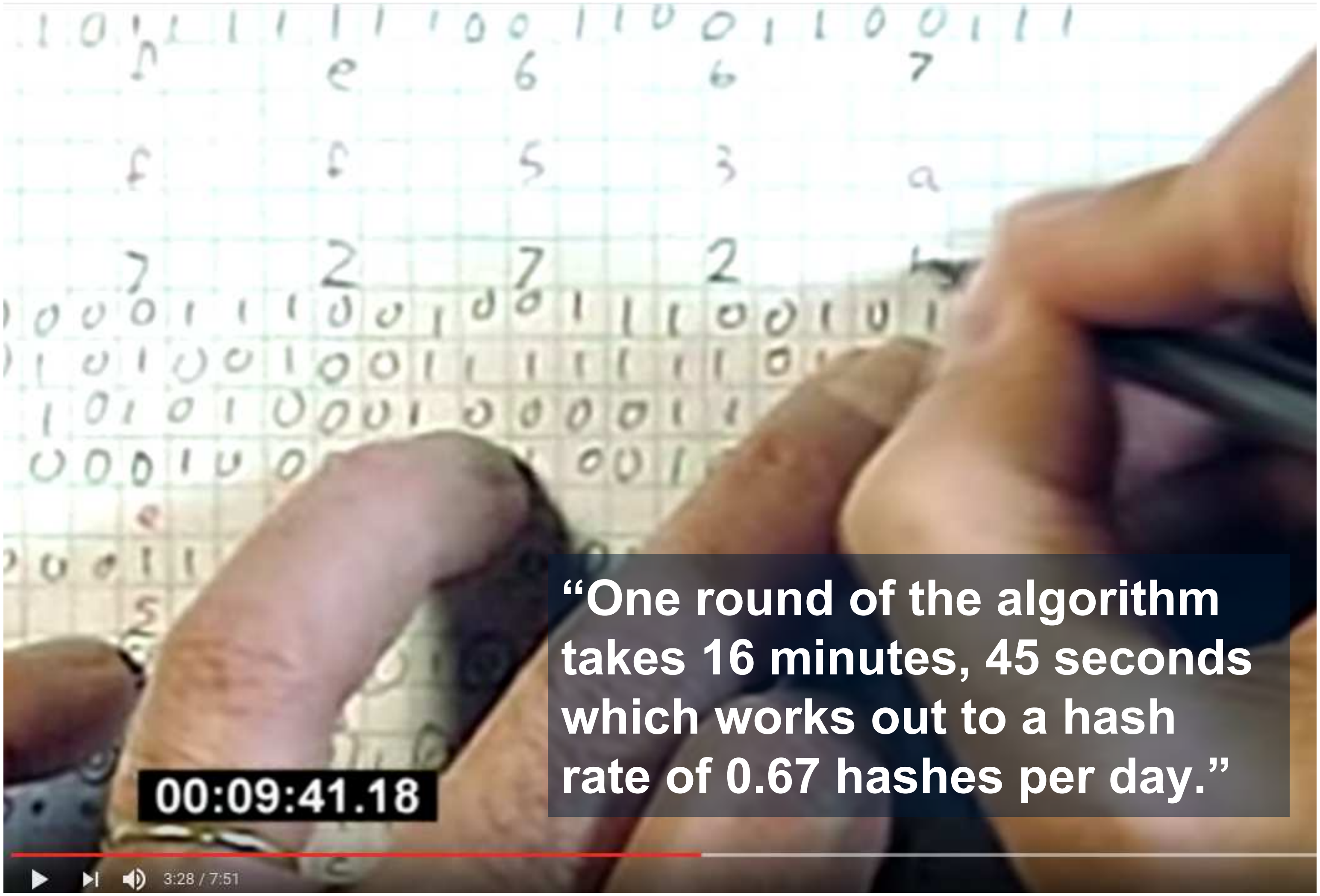
Confusion and Diffusion

The **SHA256** compression function, **h**



$$\mathrm{Ch}(E,F,G) = (E \wedge F) \oplus (\neg E \wedge G)$$
$$\mathrm{Ma}(A,B,C) = (A \wedge B) \oplus (A \wedge C) \oplus (B \wedge C)$$
$$\Sigma_0(A) = (A \ggg 2) \oplus (A \ggg 13) \oplus (A \ggg 22)$$
$$\Sigma_1(E) = (E \ggg 6) \oplus (E \ggg 11) \oplus (E \ggg 25)$$

https://www.youtube.com/watch?v=y3dqhixzGVo



Other hash functions:

MD5

Once ubiquitous

Broken in 2004

Turns out to be easy to find ***collisions***

(pairs of messages with same MD5 hash)

SHA-1

Currently widely used, but going away

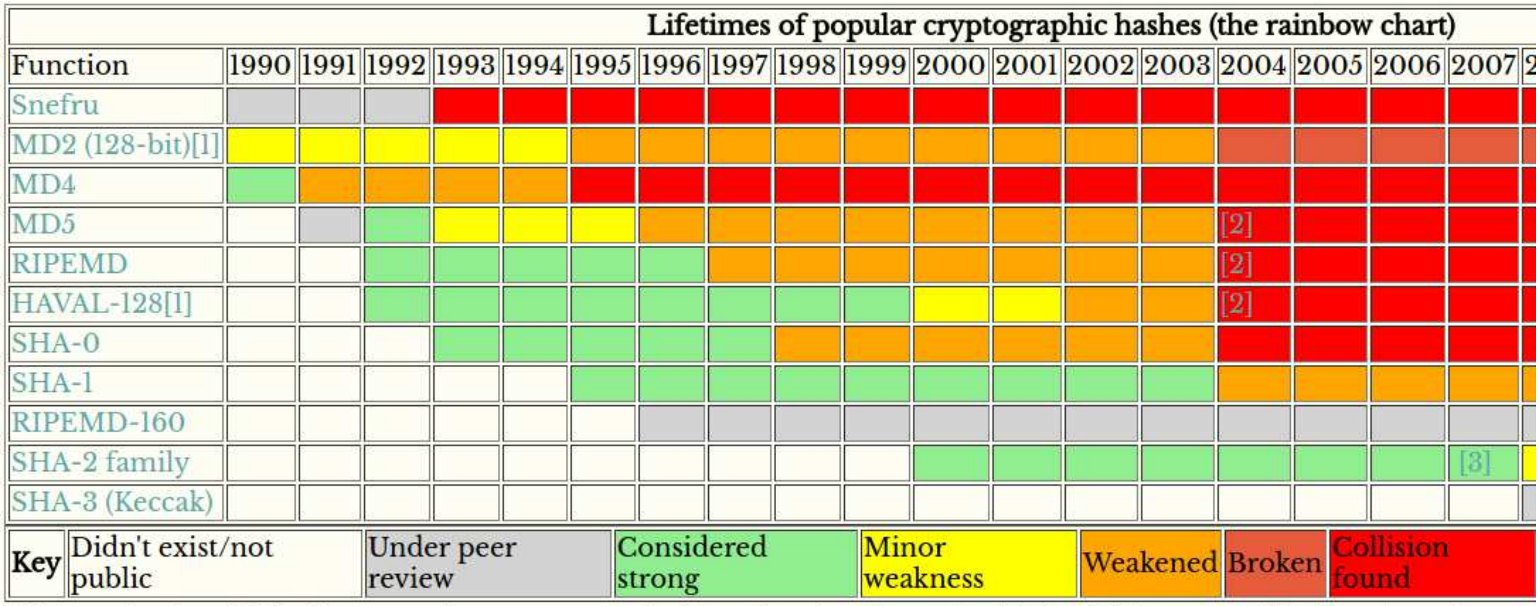
Broken in 2017

Don’t use in new applications

SHA-3

Different construction: “Sponge”

Not susceptible to ***length-extension*** http://valerieaurora.org/hash.html



[1] Note that 128-bit hashes are at best 2^64 complexity to break; using a 128-bit hash is irresponsible based on sheer digest length.

[2] What happened in 2004? Xiaoyun Wang and Dengguo Feng and Xuejia Lai and Hongbo Yu happened.

[3] In 2007, the NIST launched the SHA-3 competition because "Although there is no specific reason to believe that a practical attack on any of the SHA-2 family of hash functions is imminent, a successful collision attack on an algorithm in the SHA-2 family could have catastrophic effects for digital signatures." One year later the first strength reduction was published.

The Hash Function Lounge has an excellent list of references for most of the dates. Wikipedia now has references to the rest.

How do you find a collision?

- Pigeonhole principle: collisions must exist

Input space $\{0,1\}^*$ larger than output $\{0,1\}^{256}$

- Birthday attack: build a table with 2^{128} entries

With ~50% probability, have a collision

- Cycle finding: “Tortoise and hare” algorithm

$h(x)$, $h(h(x))$, $h(h(h(x)$, .., $h^i(x)$

- These are **generic**—actual attacks rely on **structure** of the particular function

Most cryptographic primitives come with a **security parameter**

Usually k, or λ

- Often corresponds to a key size
- Cryptography protocols run in **polynomial** time

i.e., as a function of λ, O(poly(λ))

- Ideally, we can show that the chance of failure is **negligible**, or **vanishingly** small as a function of λ

O(negl(λ))

Concrete Parameterization

How large of a digest size should we choose?

1. Estimate an attacker’s budget

E.g., the entire NSA

2. Consider the best known attacks

Reduction from protocol to well-studied problem

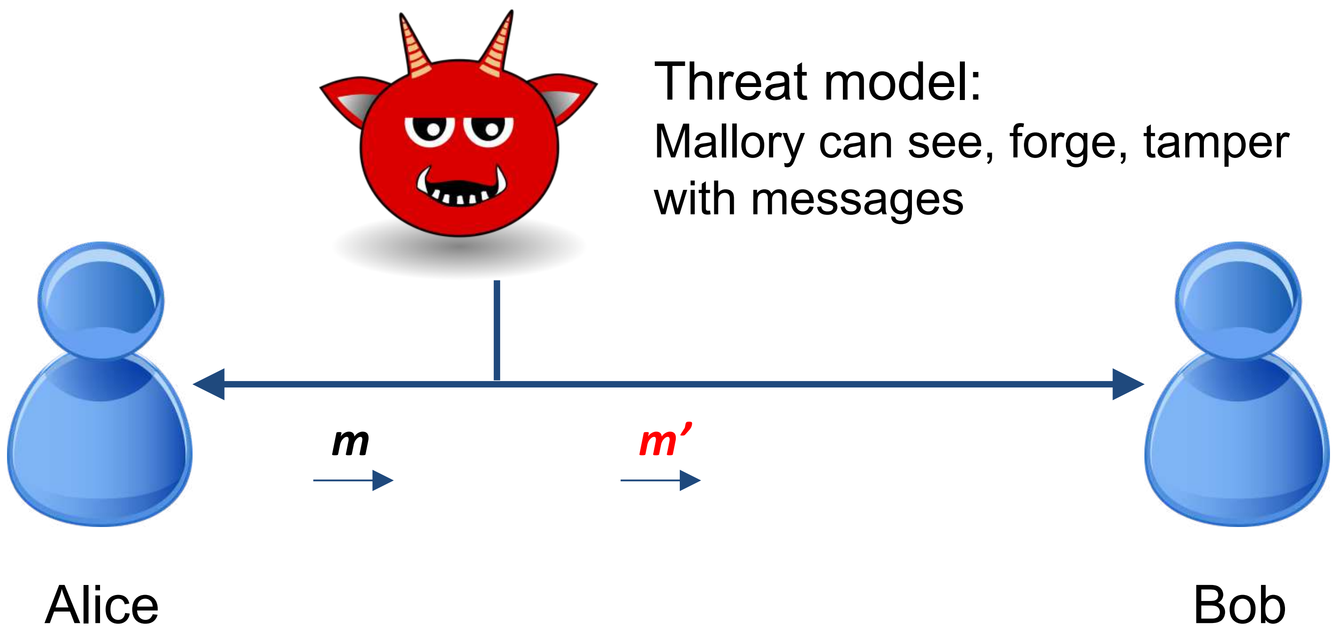
3. Add a safety margin

If all goes well, adding 1 bit increases search space by 2x

Goal: Message Integrity

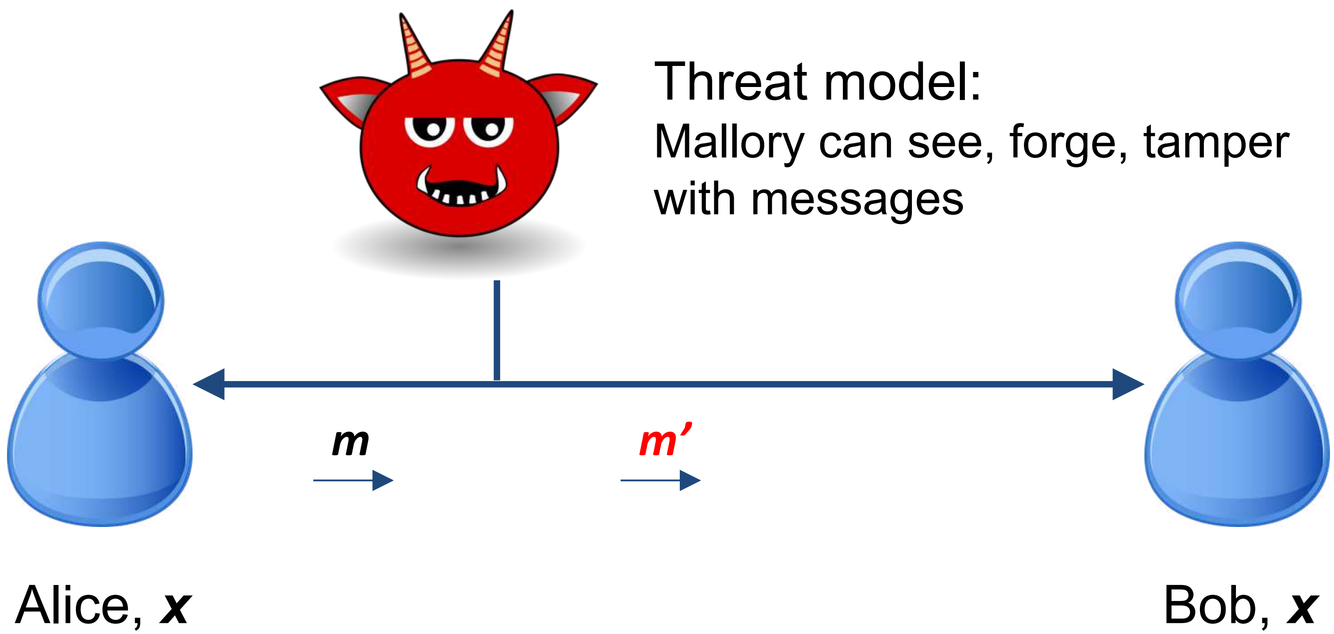
Alice wants to send message ***m*** to Bob
Mallory wants to trick Bob into accepting a message Alice didn't send

Threat model:
Mallory can see, forge, tamper with messages



Goal: Message Integrity

Alice wants to send message ***m*** to Bob
Mallory wants to trick Bob into accepting a message Alice didn't send



Setup assumption: ***shared secret***

Solution: Message Authentication Code (MAC)

1. Alice computes **v** := ***f***(**m**)

2.

3. Bob verifies that **v′** = ***f***(**m′**), accepts message iff this is true

Function ***f*** ?

Easily computable by Alice and Bob; not computable by Mallory

(Idea: Secret only Alice & Bob know)

We’re sunk if Mallory can learn ***f***(**m’**) for any **m** ≠ **m’**!

Candidate ***f***:

Random function

Input: Any size up to huge maximum

Output: Fixed size (e.g. 256 bits)

Defined by a giant lookup table that's filled in by flipping coins

0	→	0011111001010001…
1	→	1110011010010100…
2	→	0101010001010000…
⋮		⋮

Completely <u>impractical</u>

[Why?]

Provably <u>secure</u>

[Why?]

Want a function that's practical but "looks random"…

Pseudorandom function (PRF)

Let's build one:

Start with a big *family of functions*

$\boldsymbol{f}_0, \boldsymbol{f}_1, \boldsymbol{f}_2, \ldots$ all known to Mallory

Use $\boldsymbol{f}_\mathbf{k}$, where **k** is a secret value
(or "key") known only to Alice/Bob

k is (say) 256 bits, chosen randomly

Kerckhoffs's Principle [Why?]

Don't rely on secret functions

Use a secret key, to choose from a function family

More formal definition of a secure **PRF**:

Game against Mallory

1. We flip a coin secretly to get bit **b**
2. If **b**=0, let ***g*** be a random function
   If **b**=1, let ***g*** = $\boldsymbol{f}_{\mathbf{k}}$, where **k** is a randomly chosen secret
3. Repeat until Mallory says "stop":
   Mallory chooses **x**; we announce ***g***(**x**)
4. Mallory guesses **b**

We say ***f*** is a *secure PRF* if Mallory can't do better than random guessing*

i.e., $\boldsymbol{f}_{\mathrm{k}}$ is indistinguishable in practice from a random function, unless you know k

Important fact: There's an algorithm that always wins for Mallory

[What is it?] [How to fix it?]

A solution for Alice and Bob:

1. Let $\boldsymbol{f}$ by a secure PRF
2. In advance, choose a random **k** known only to Alice and Bob
3. Alice computes $\mathbf{v} := \boldsymbol{f}_k(\mathbf{m})$

4. Bob verifies that $\mathbf{v'} = \boldsymbol{f}_k(\mathbf{m'})$,
   accepts message iff this is true

[Important assumptions?]

What if Alice and Bob want to send more than one message?

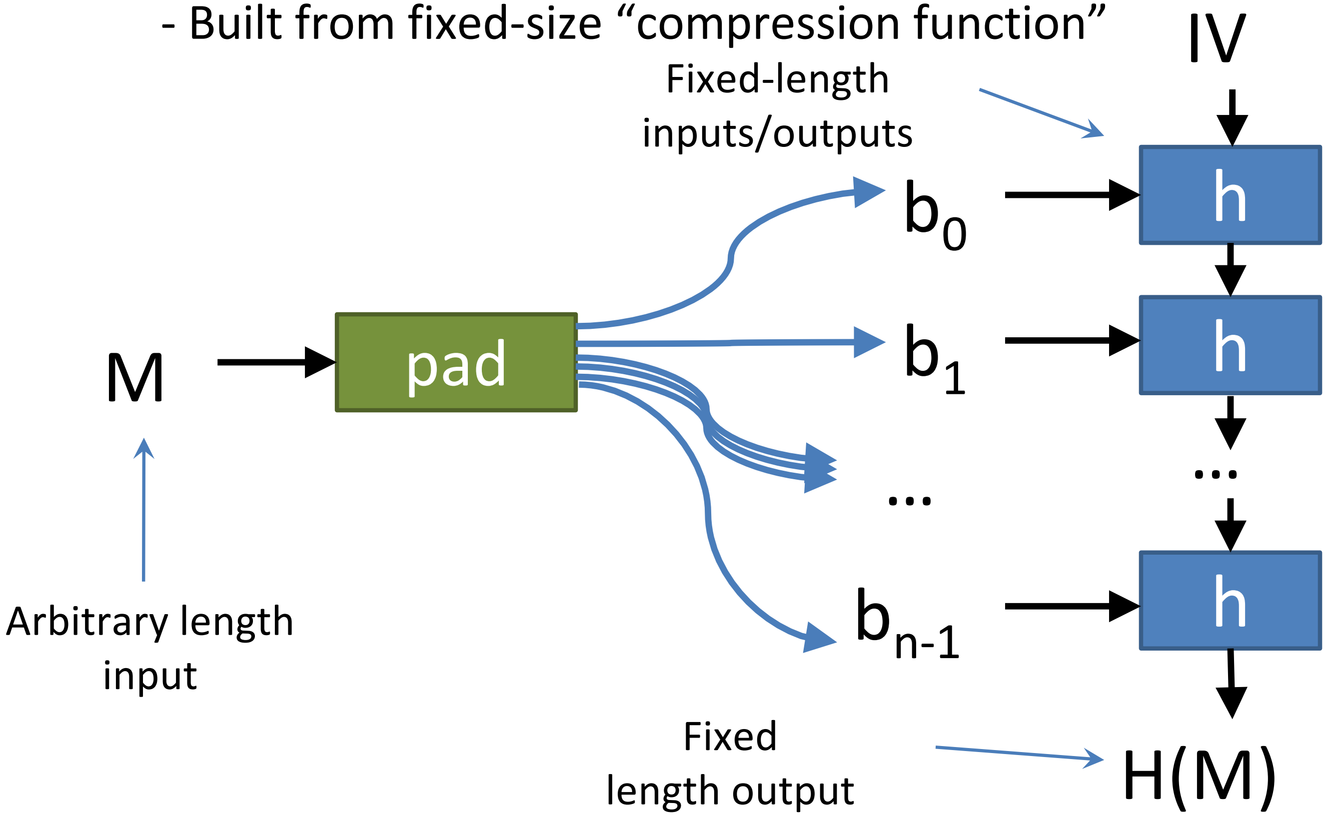
[Attacks?]   [Solutions?]

Is this a secure PRF?

$$\boldsymbol{f}_{\mathbf{k}}(\mathbf{m}) = \mathrm{SHA256}(\ \mathbf{k} \ || \ \mathbf{m}\ )$$

Merkle–Damgård Construction

- Arbitrary-length input
- Fixed-length output
- Built from fixed-size “compression function”



Recommended Approach: Hash-based MAC (HMAC)

HMAC-SHA256

see RFC 2104

HMAC$_{\mathbf{k}}$($\mathbf{m}$) =

$$\text{SHA256}\left(k \oplus c_1 \parallel \text{SHA256}\left(k \oplus c_2 \parallel m\right)\right)$$

XOR

`0x3636…`

Concatenation

`0x5c5c…`

SHA256 function

takes arbitrary length input,

returns 256-bit output

Message Authentication Code (MAC)
e.g. HMAC-SHA256
vs.

Cryptographic hash function
e.g. SHA256
not a strong PRF

Used to think the distinction didn't matter, now we think it does

e.g., ***length extension attacks***

Better to use a MAC/PRF (not a hash)

```
$ openssl dgst -sha256 -hmac <key>
```

MAC Crypto Game

Game against Mallory

1. Give Mallory MAC(k, m_i) for all m_i in M
In other words, Mallory has an ***oracle***
Mallory can choose next m_i after seeing answer

2. Mallory tries to discover MAC(k, m’) for a new m’ not in M

We can show the **MAC game** ***reduces*** to the **PRF game**. Mallory wins MAC game → she wins PRF game.

This is a **Security Proof**

What is a **Security Proof**?

- A ***reduction*** from an ***attack on your protocol*** to an attack on a ***widely studied, hard problem***
- Excludes large classes of attacks, guides **composition**
    - Proofs are in **models**. So, attack outside the model!
- It does **NOT** ***prove*** that your protocol is ***secure***
- We don't know if there are any hard problems!
- The field of **Modern Cryptography** is based on proofs
- Most widely used primitives (SHA-256, AES, DSA) have no security proof. We rely on them because they're widely studied

So Far

Message Integrity

Next time …

The classic problem in crypto:

How can Alice send Bob a message, with **confidentiality**?