

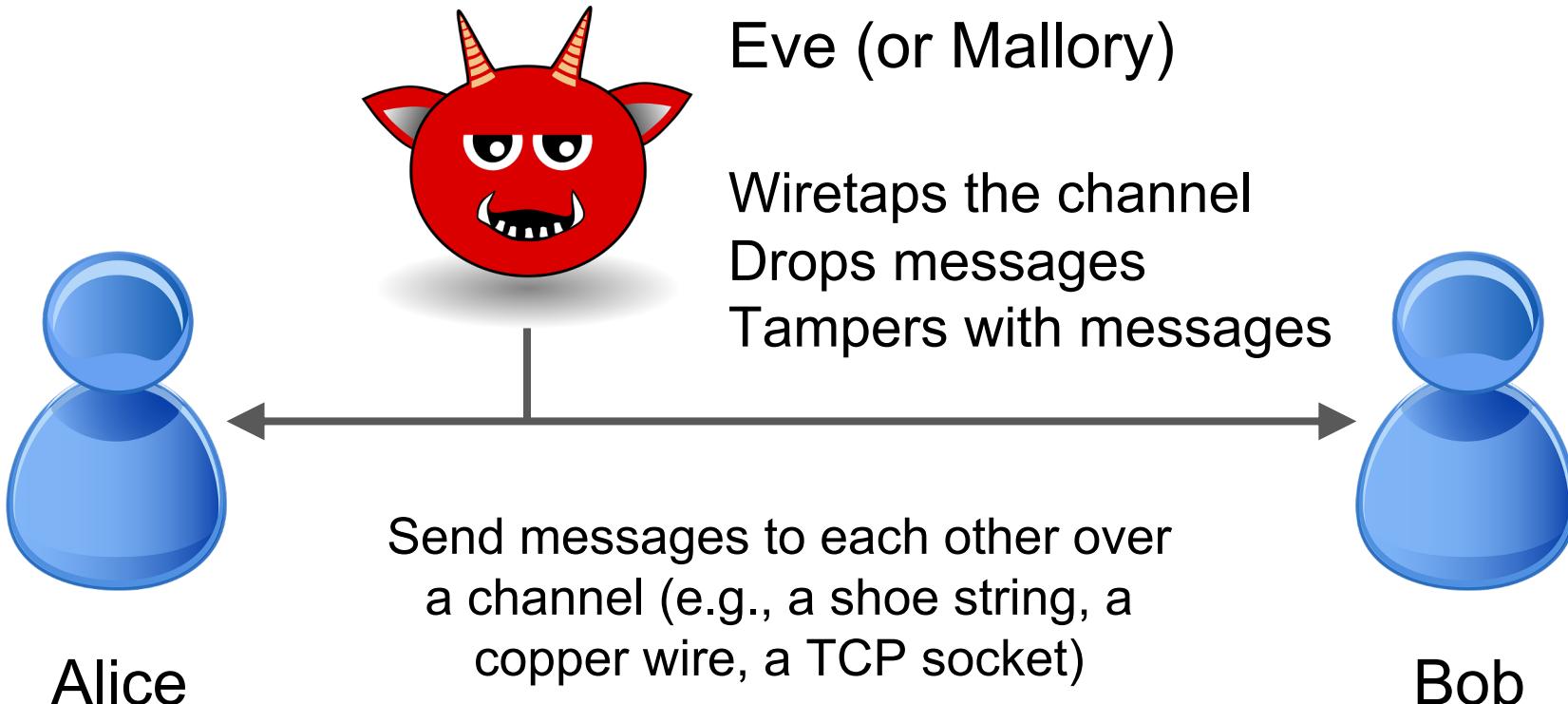
Lecture 28 – Message Integrity

Stephen Checkoway

Oberlin College

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:

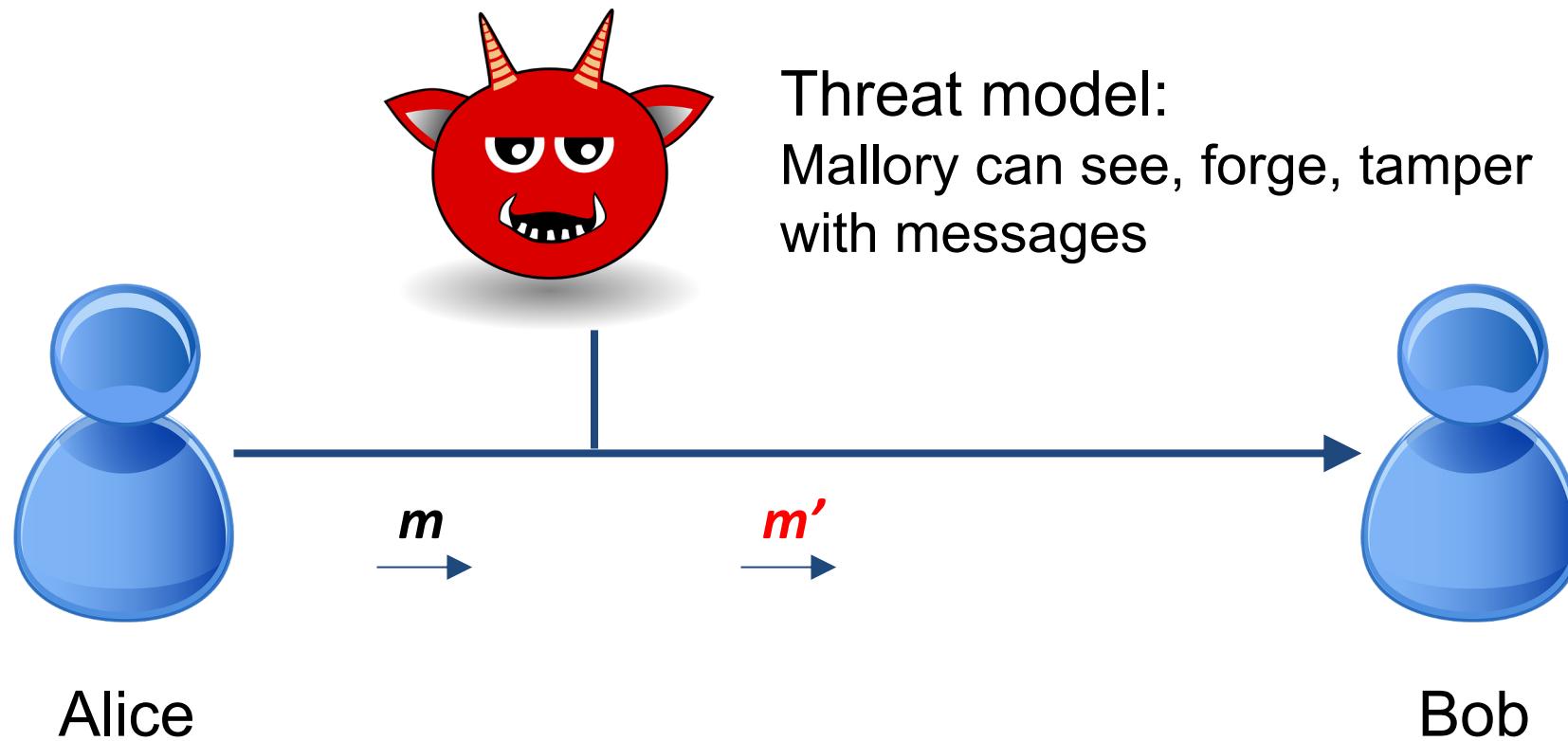
- Confidentiality: secrecy, privacy
 - Integrity: tamper resilience
 - Availability
 - Non-repudiability, or deniability
- many more properties

Message Integrity

Hashes, MACs

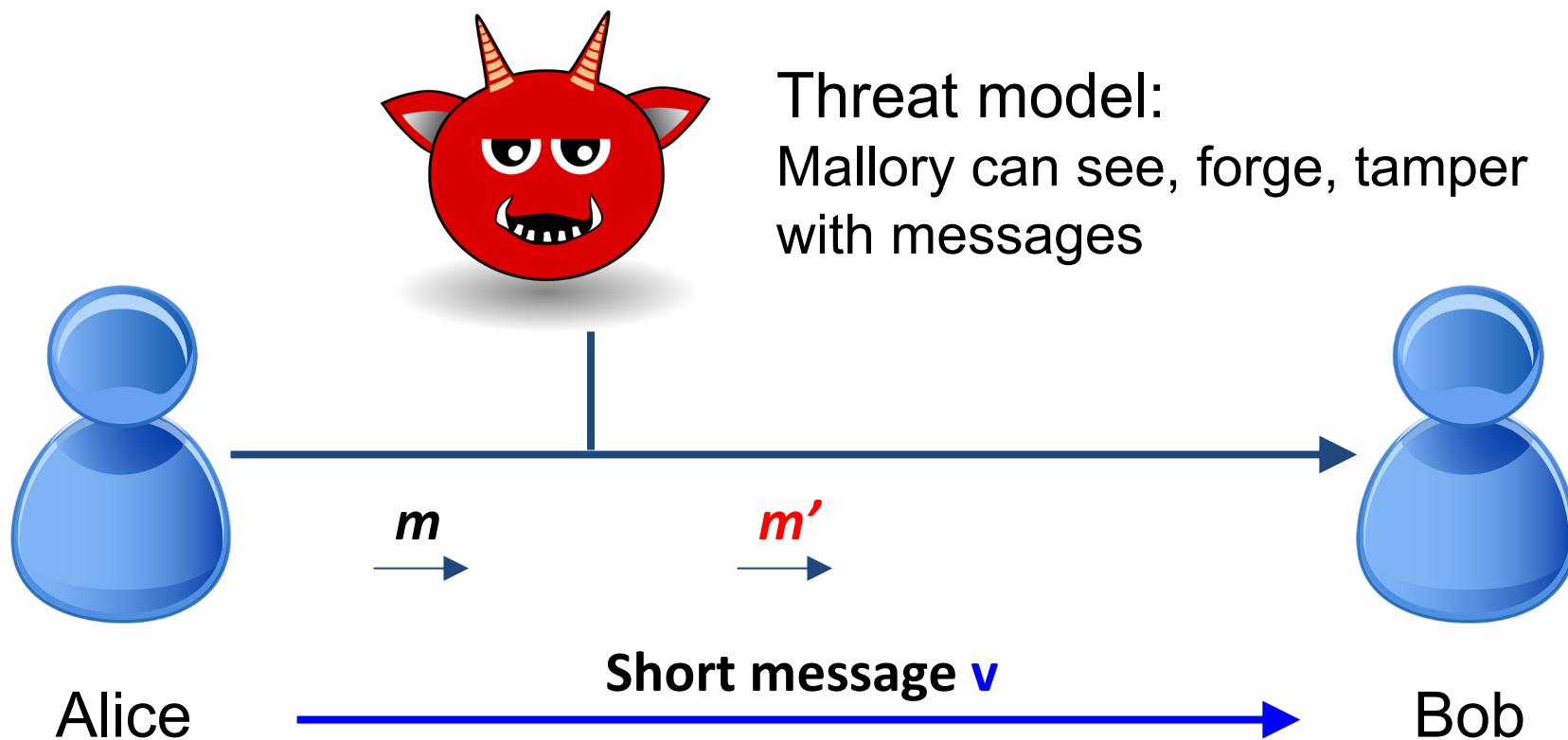
Goal: Secure File Transfer

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



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

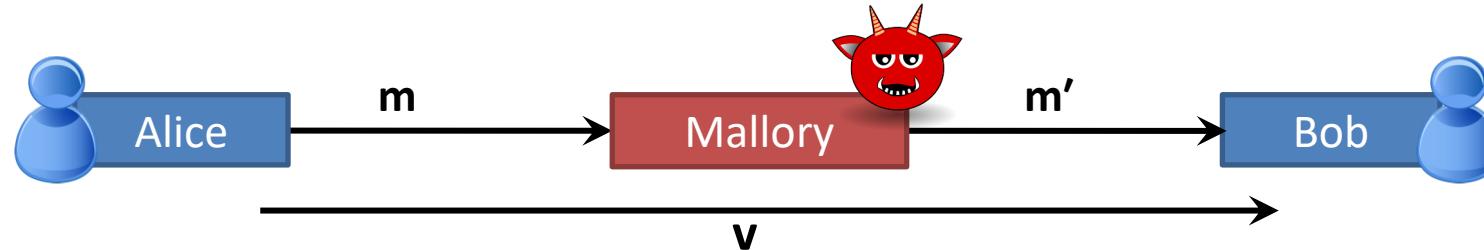


Setup assumption: *Securely transfer a short message!*

Solution: Collision Resistant Hash Function (CRHF)

Hash Function $h: \{0,1\}^* \rightarrow \{0,1\}^{256}$ (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 $m' \neq m$
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

First pre-image resistance:

Which of these properties implies which others?

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:

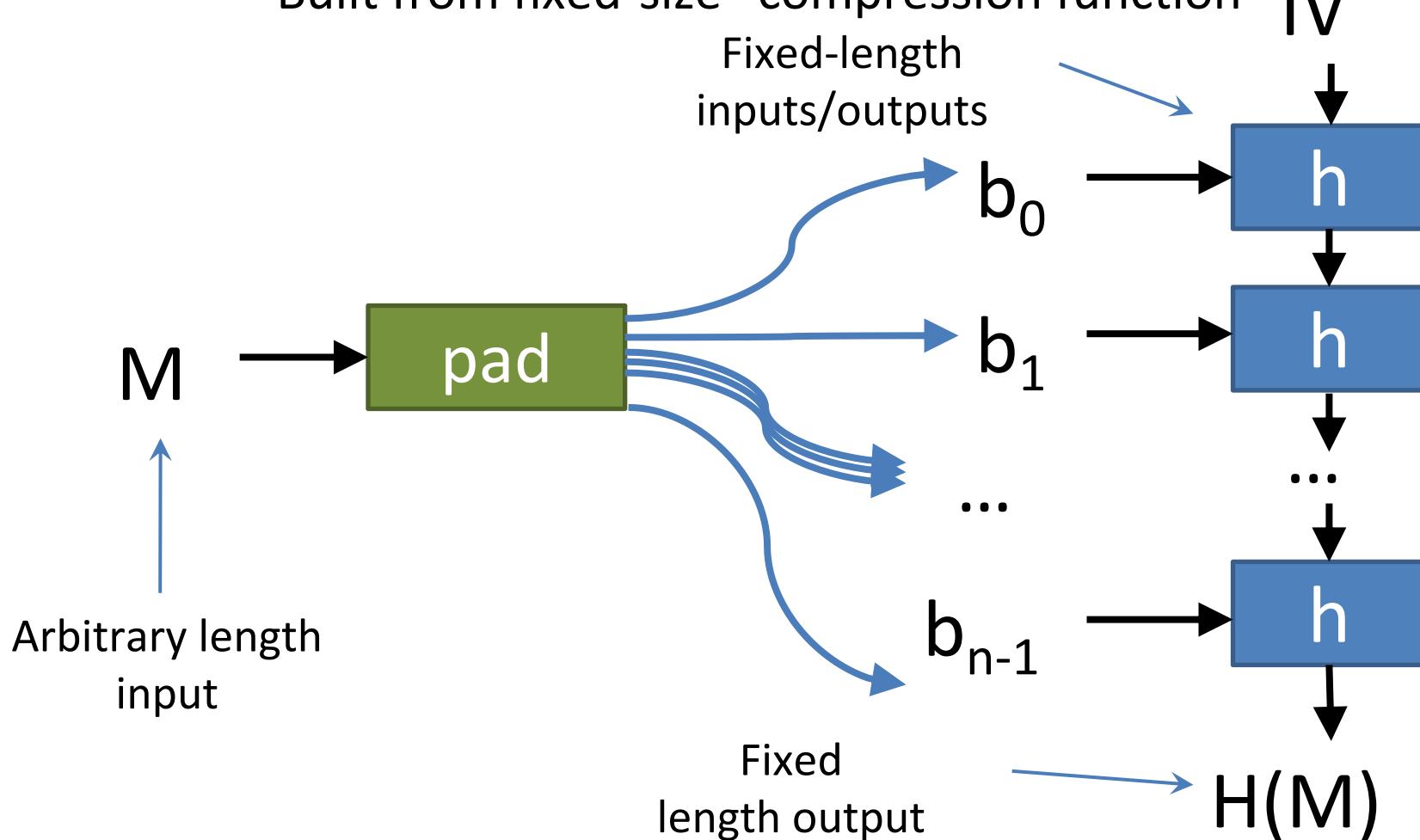
It is computationally infeasible to find *any* $m_1 \neq m_2$ s.t. $h(m_1) = h(m_2)$

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

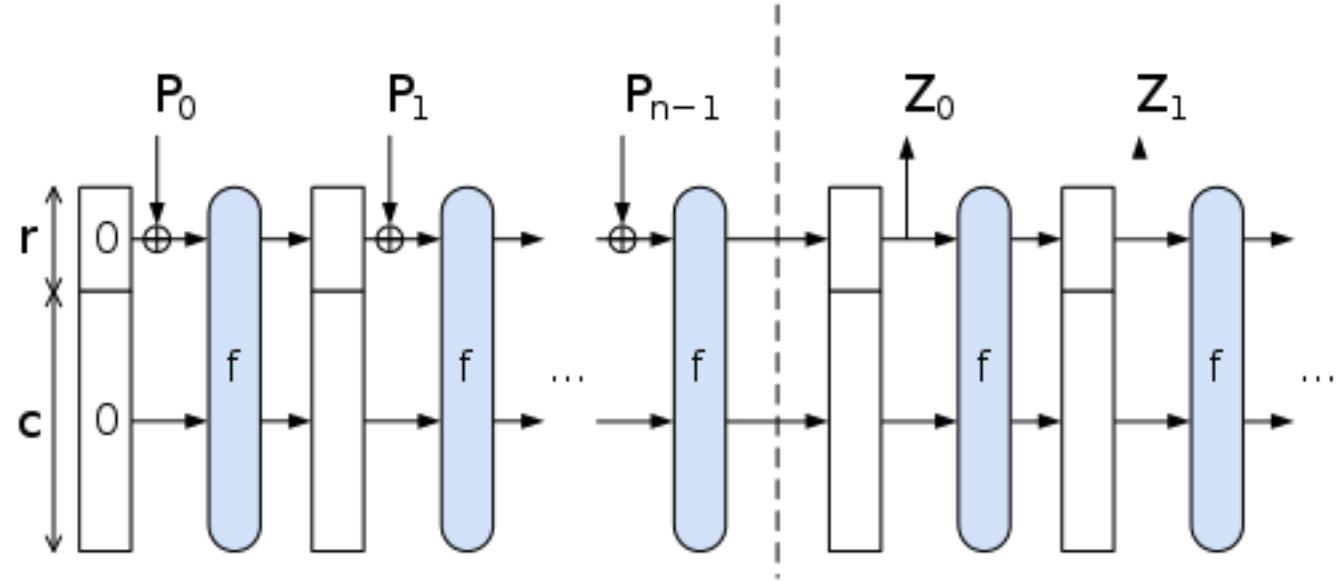
Merkle–Damgård Construction

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



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

The **SHA256** compression function, h

Cryptographic hash

Input: arbitrary length data
(No key)

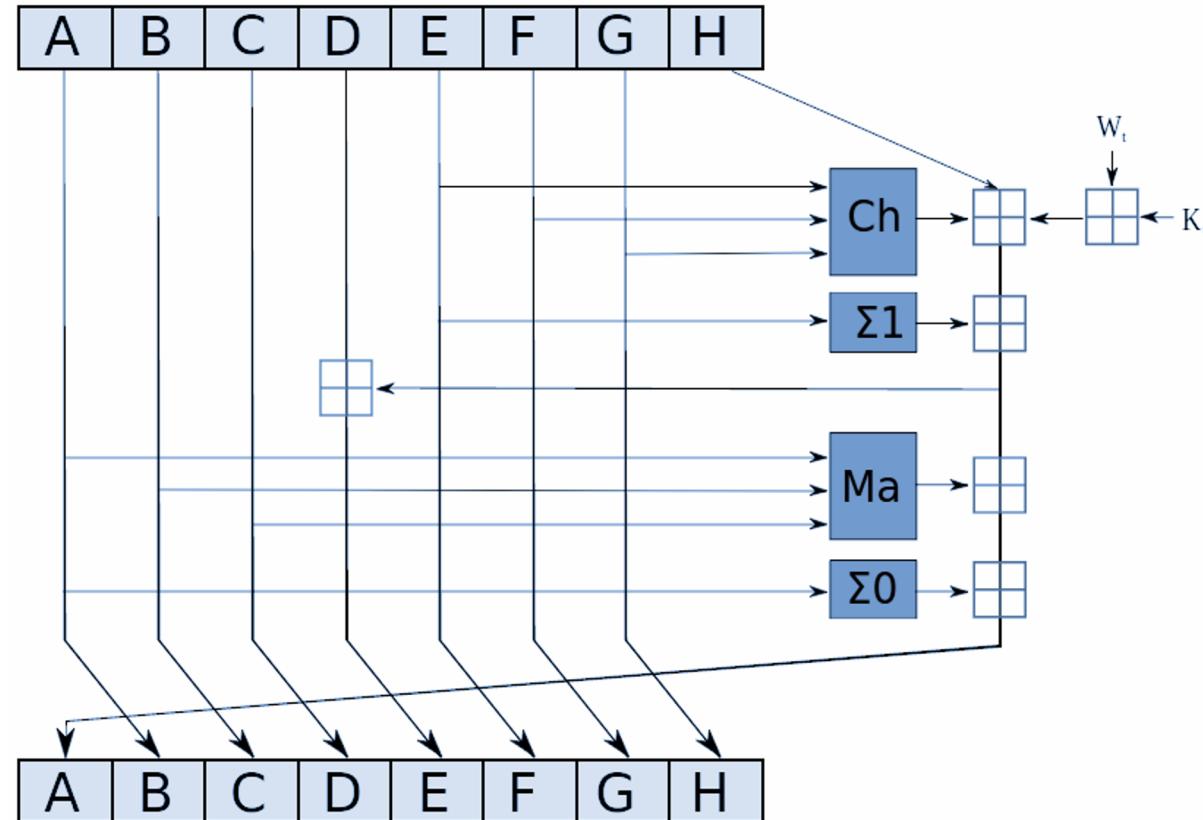
Output: 256 bits

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***

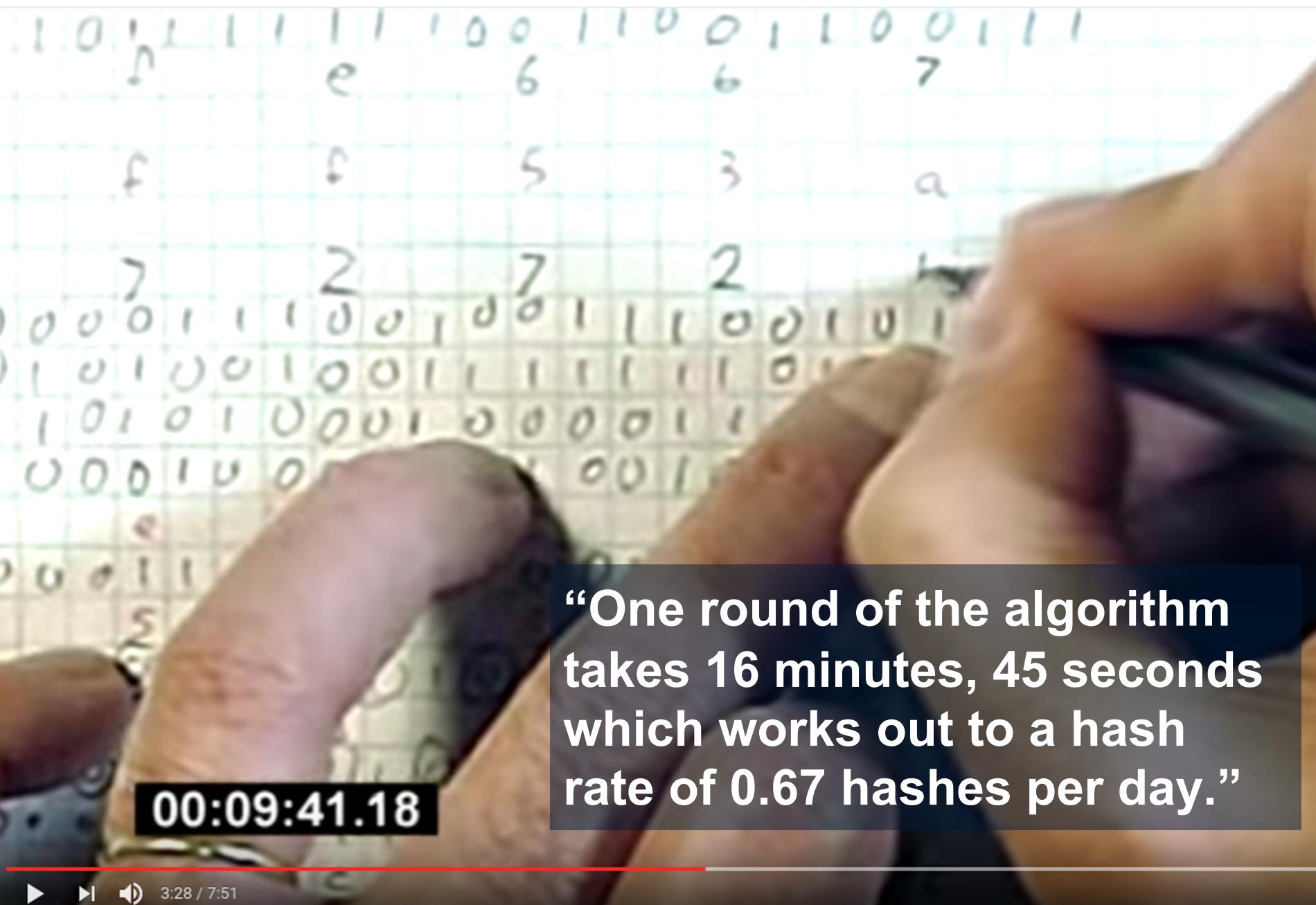


$$Ch(E, F, G) = (E \wedge F) \oplus (\neg E \wedge G)$$

$$Ma(A, B, C) = (A \wedge B) \oplus (A \wedge C) \oplus (B \wedge C)$$

$$\Sigma_0(A) = (A \ggg 2) \oplus (A \ggg 13) \oplus (A \ggg 22)$$

$$\Sigma_1(E) = (E \ggg 6) \oplus (E \ggg 11) \oplus (E \ggg 25)$$



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

Still in use in some places, but going away fast

Broken in 2017

Don't use in new applications

SHA-3

Different construction: "Sponge"

Not susceptible to ***length-extension***

Lifetimes of popular cryptographic hashes (the rainbow chart)																												
Function	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017
Snefru	Grey	Grey	Grey	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	
MD2 (128-bit)[1]	Yellow	Yellow	Yellow	Yellow	Orange	Orange	Orange	Orange	Orange	Orange	Orange	Orange	Orange	Orange	Orange	Orange	Orange	Orange	Orange	Orange	Orange	Orange	Orange	Orange	Orange	Orange	Orange	
MD4	Green	Orange	Orange	Orange	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	
MD5		Grey	Green	Yellow	Yellow	Orange	Orange	Orange	Orange	Orange	Orange	Orange	Orange	[2]	Red													
RIPEMD			Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	[2]	Red													
HAVAL-128[1]			Green	Green	Green	Green	Green	Green	Green	Green	Yellow	Yellow	Orange	[2]	Red													
SHA-0			Green	Green	Green	Green	Green	Green	Green	Green	Orange	Orange	Orange	Orange	Red													
SHA-1				Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	[3]	Green
RIPEMD-160																												Yellow
SHA-2 family																			[4]	Yellow								
SHA-3 (Keccak)																												
Key	Didn't exist/not public	Under peer review	Considered strong	Minor weakness	Weakened	Broken	Collision found																					

[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] Google spent [6500 CPU years and 110 GPU years](#) to convince everyone we need to stop using SHA-1 for security critical applications. Also because it was cool.

[4] 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.

Important use of Collision-Resistant Hash Functions: *Commit names of files in source code repositories*

- The “name” of a commit is a hash of all the files in its contents.
- The hash of a folder is the hash of the hashes of each file contained within
 - i.e. an “authenticated data structure”
- [What are the advantages of this?]
- [What happens if there is a collision? 1st preimage?]

RISK ASSESSMENT —

Watershed SHA1 collision just broke the WebKit repository, others may follow

"Please exercise care" with colliding PDFs, researchers advise software developers.

DAN GOODIN - 2/24/2017, 2:28 PM



“

Yes - please exercise care, as SHA-1 colliding files are currently breaking SVN repositories. Subversion servers use SHA-1 for deduplication and repositories become corrupted when two colliding files are committed to the repository. This has been discovered in WebKit's Subversion repository and independently confirmed by us. Due to the corruption the Subversion server will not accept further commits.



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

$h(x), h(h(x)), h(h(h(x))), \dots, h^i(x)$

- 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 , $O(\text{poly}(\lambda))$
- Ideally, we can show that the chance of failure is **negligible**, or **vanishingly** small as a function of λ
 $O(\text{negl}(\lambda))$

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

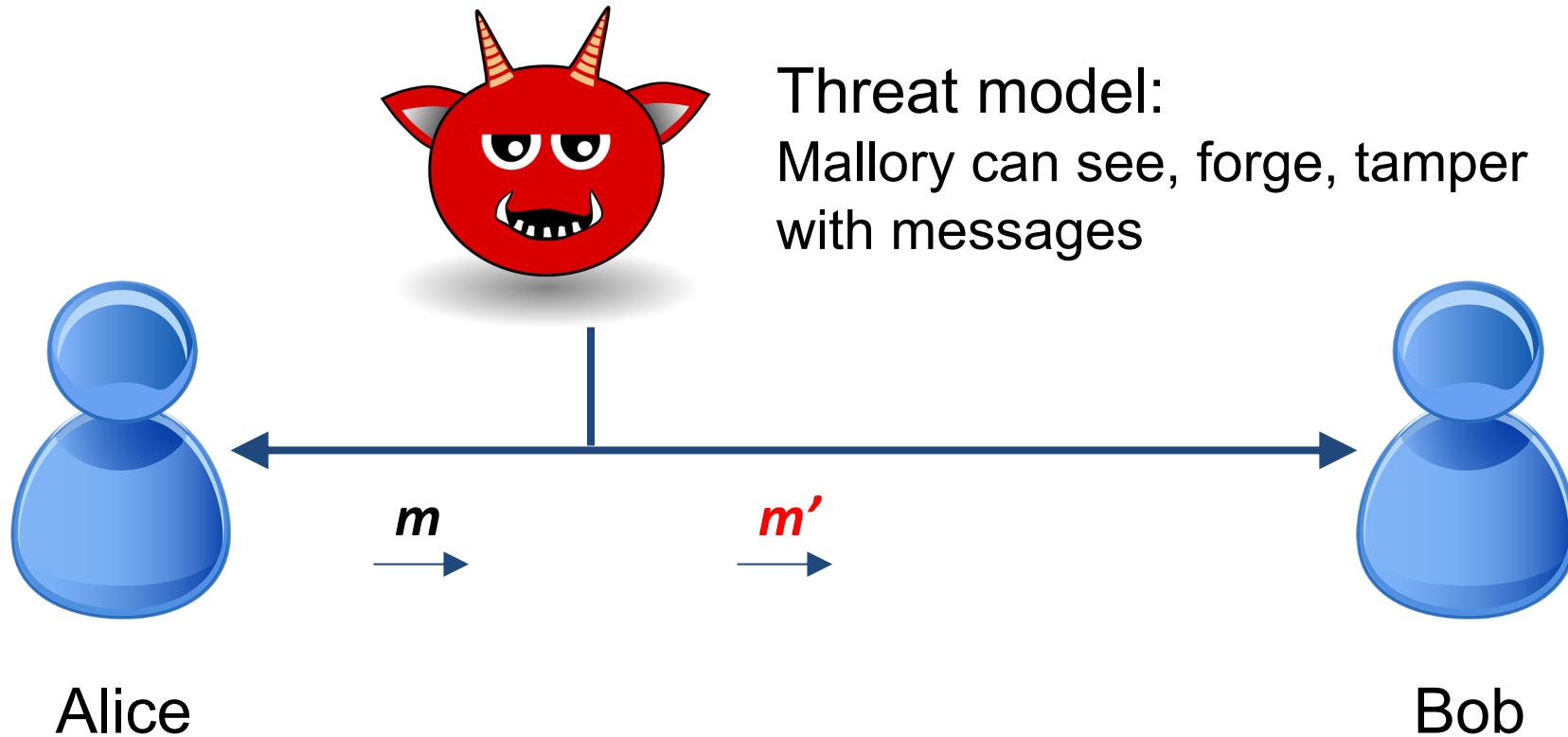
3. Add a safety margin

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

Goal: Message Integrity

Alice wants to send message m to Bob

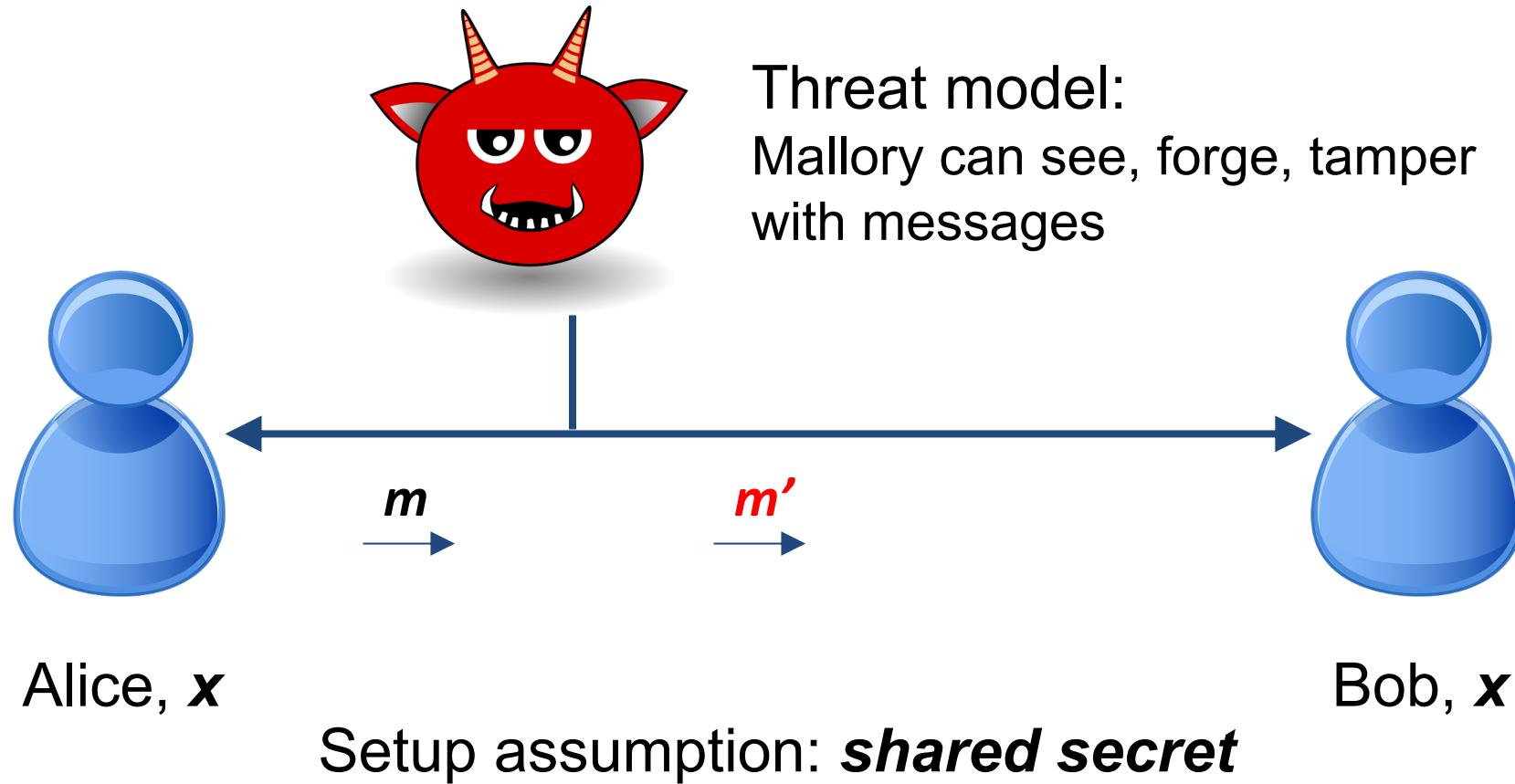
Mallory wants to trick Bob into accepting a message Alice didn't send



Goal: Message Integrity

Alice wants to send message m to Bob

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



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 $m \neq m'$!

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 \rightarrow 0011111001010001...

1 \rightarrow 1110011010010100...

2 \rightarrow 0101010001010000...

Completely impractical

[Why?]

Provably secure

[Why?]

Want a function that's practical but “looks random”...

Pseudorandom function (PRF)

Let's build one:

Start with a big *family of functions*

f_0, f_1, f_2, \dots all known to Mallory

Use f_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 \mathbf{b}
2. If $\mathbf{b}=0$, let \mathbf{g} be a random function
If $\mathbf{b}=1$, let $\mathbf{g} = \mathbf{f}_k$, where k is a randomly chosen secret
3. Repeat until Mallory says “stop”:
Mallory chooses x ; we announce $\mathbf{g}(x)$
4. Mallory guesses \mathbf{b}

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

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



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

[Important assumptions?]

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

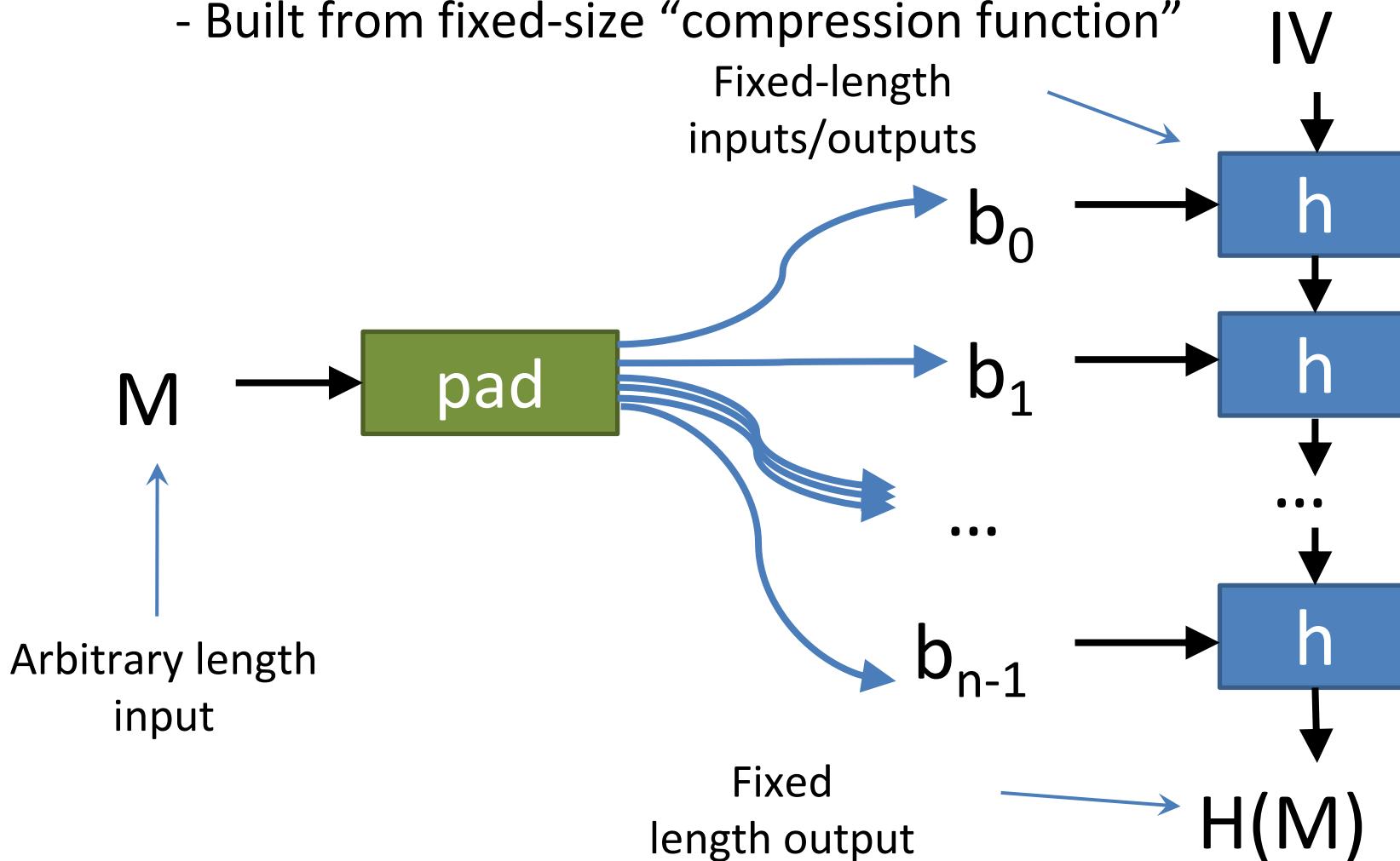
[Attacks?] [Solutions?]

Is this a secure PRF?

$$f_k(m) = \text{SHA256}(k \parallel m)$$

Merkle–Damgård Construction

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



Formalize SHA-256

- Let IV be the initialization vector
- Let $h: \{0,1\}^{256} \times \{0,1\}^{512} \rightarrow \{0,1\}^{256}$ be the compression function
- Let $\text{pad}(len)$ be the padding appended to a len -bit message
- $msg \parallel \text{pad}(|msg|) = B_0 \parallel B_1 \parallel \dots \parallel B_{n-1}$ where $|B_i| = 512$ bits
- Define $C_0 = \text{IV}$ and $C_i = h(C_{i-1}, B_{i-1})$ for $i > 0$
- Then $\text{SHA256}(msg) = C_n$

Formalizing our proposed PRF

- $msg \parallel \text{pad}(|msg|) = B_0 \parallel B_1 \parallel \dots \parallel B_{n-1}$ where $|B_i| = 512$ bits
- Define $C_0 = \text{IV}$ and $C_i = h(C_{i-1}, B_{i-1})$ for $i > 0$
- Then $\text{SHA256}(msg) = C_n$
- If $msg = k \parallel m$, then $f_k(m) = \text{SHA256}(k \parallel m) = C_n$

Length extension attack

- $msg \parallel \text{pad}(|msg|) = B_0 \parallel B_1 \parallel \cdots \parallel B_{n-1}$ where $|B_i| = 512$ bits
- Define $C_0 = \text{IV}$ and $C_i = h(C_{i-1}, B_{i-1})$ for $i > 0$
- If $msg = k \parallel m$, then $f_k(m) = \text{SHA256}(k \parallel m) = C_n$
- Consider $msg' = k \parallel m \parallel \text{pad}(|k| + |m|) \parallel m'$
- $msg' \parallel \text{pad}(|msg'|) = B_0 \parallel B_1 \parallel \cdots \parallel B_{n-1} \parallel B_n \parallel \cdots \parallel B_{n+j-1}$
- Then $f_k(m \parallel \text{pad}(|k| + |m|) \parallel m') = C_{n+j}$
- But $C_{n+j} = h(C_{n+j-1}, B_{n+j-1}) = \cdots = h(h(\cdots h(C_n, B_n), \cdots), B_{n+j-1})$

Recommended Approach: Hash-based MAC (HMAC)

HMAC-SHA256

see RFC 2104

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

The diagram illustrates the construction of HMAC-SHA256. It shows two nested SHA256 functions. The inner function takes a key k , a constant c_2 (represented by the hex value 0x5c5c...), and a message m as input. The output of this inner function is then concatenated (indicated by a double vertical bar ||) with the result of an XOR operation between the key k and another constant c_1 (represented by the hex value 0x3636...). This final result is passed as input to the outer SHA256 function, which also takes the key k and constant c_1 as input.

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 $\text{MAC}(k, m_i)$ 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 $\text{MAC}(k, m')$ for a new m' not in M

We can show the **MAC game reduces** to the **PRF game**. Mallory wins MAC game \rightarrow 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**?