

---

# **Introduction to Cryptography and Security Mechanisms:**

## **Unit 10**

### **Data integrity**

# Learning Outcomes

---

- Appreciate that there are different levels of data integrity
- Identify the different properties of a hash function
- Comment on different applications of a hash function and which properties they require
- Explain how to use a MAC to provide data origin authentication
- Describe two different approaches to constructing a MAC
- Compare different ways of combining MACs with encryption to provide both confidentiality and data origin authentication

# Sections

---

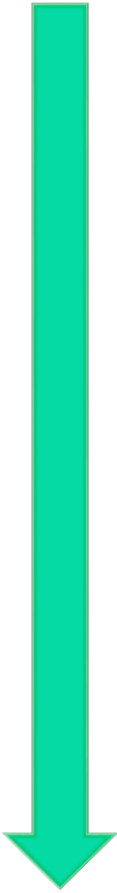
1. Hash functions
2. Message authentication codes
3. Authenticated encryption

---

# 1. Hash functions

# Levels of data integrity

---

- 
- Protection against accidental errors
    - CRC checks
  - Protection against simple manipulations
    - Hash functions
  - Protection against active attacks
    - MACs
  - Protection against repudiation attacks
    - Digital signatures

# Uses of hash functions

---

**Hash functions** have many important and varied uses:

- As strong one-way functions
  - Hash functions are sometimes used to store highly confidential data such as passwords
- To provide a weak notion of data integrity
  - Hash functions are often referred to as **manipulation detection codes** (or **modification detection codes**)
- As components to build other cryptographic primitives
  - Hash functions are a critical component of digital signatures with appendix
- As a means of binding data
  - Hash functions are often used within cryptographic protocols to bind data together
- As sources of pseudorandomness
  - Hash functions are used to generate cryptographic keys

# What is a hash function?

---

A **hash function** is a mathematical function which (generally):

- does not have a key and is thus publicly computable.
- has two practical properties
- has three security properties

# Practical property 1

---

**Condenses arbitrary long inputs into a fixed length output**

- No matter how long the input, the output (or **hash** or **digest**) is always the same length.
- The hash is much smaller than the input (a hash function is a **compression function**).
- We refer to an **n-bit hash function** if the hash is n bits long.



# Practical property 2

---

## Easy to compute

- A hash function should run in polynomial time
- Hash functions are expected to be faster than symmetric encryption

# Security property 1

---

The hash function should be a **one-way function**:

i.e. hash function  $h(x)$  should be:

<b>Easy to compute</b>	Given $x$ it is easy to compute $h(x)$
<b>Hard to reverse</b>	Given $h(x)$ it is hard to determine $x$

This property is often referred to as **pre-image resistance**.

# Security property 2

---

The hash function should have **second preimage resistance**:

i.e. hash function  $h(x)$  should be such that:

<b>Given a message and its hash, it is hard to find a different message with that same hash</b>	Given $x$ and $h(x)$ it is hard to find $y$ (different from $x$ ) such that $h(x)=h(y)$
---	---

# Security property 3

---

The hash function should be **collision-resistant**:

i.e. hash function  $h(x)$  should be such that:

<b>It is hard to find any two messages with the same hash</b>	It is hard to find $x$ and $y$ ( $y$ different from $x$ ) such that $h(x)=h(y)$
---	---

# Second preimages and collisions

---

- **Given a message and its hash, many second preimages must exist:**

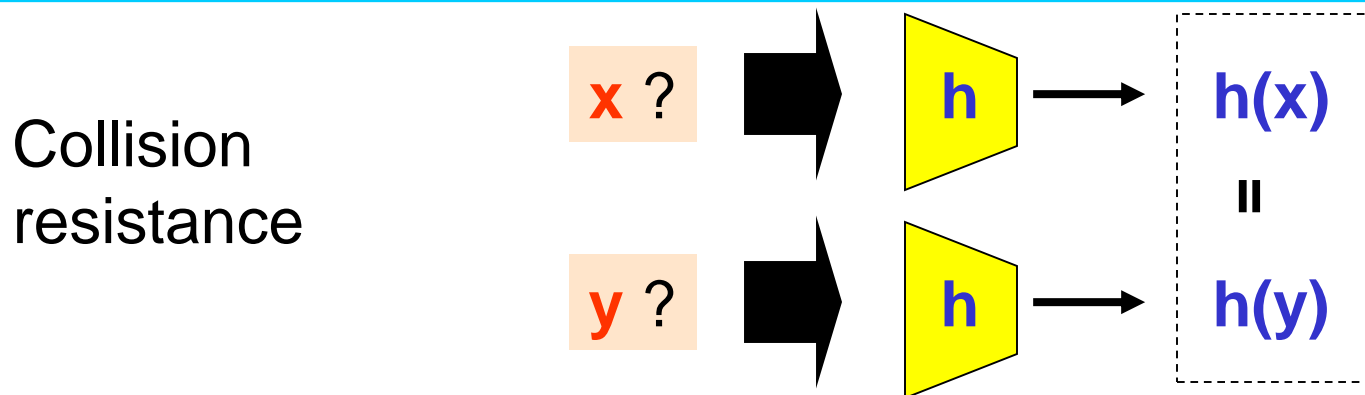
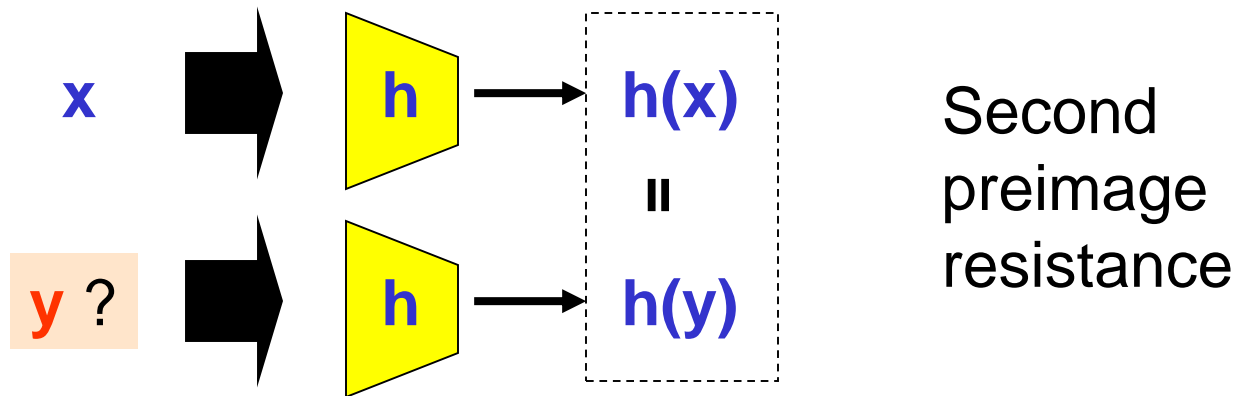
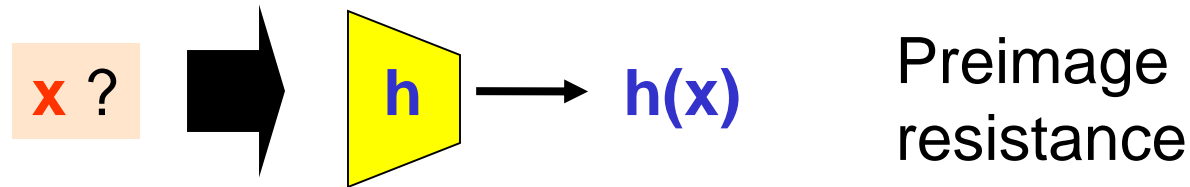
*With 60 million bank users, there must be a lot of people with the same four-digit PIN as you!*

- **It is impossible for a hash function **not** to have collisions:**

*It is impossible to give each of 60 million people a different four-digit PIN!*

**Second preimages and collisions should be **hard to find**.**

# Security property summary



# Relationship between security properties

---



**Which of these security properties is stronger?**

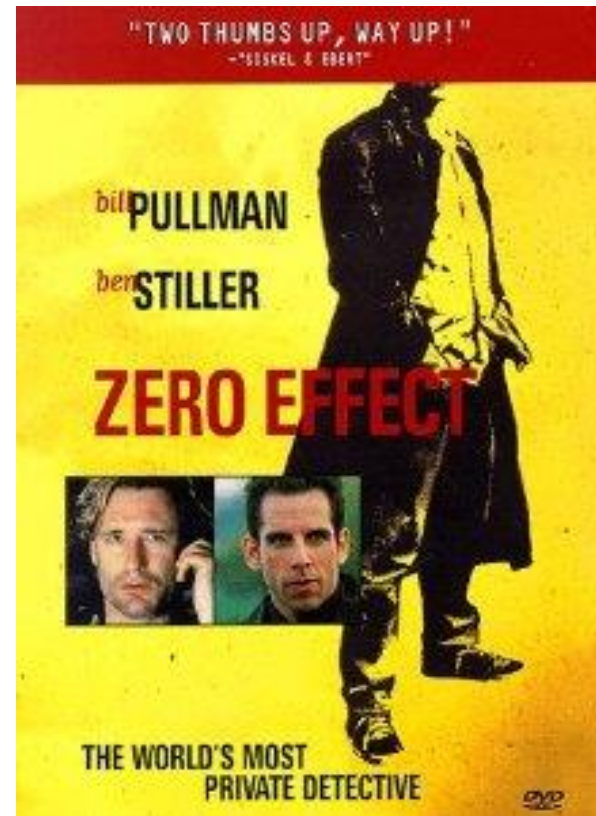
(In other words – which attacks against hash functions are “easier”?)

# Some wisdom

---

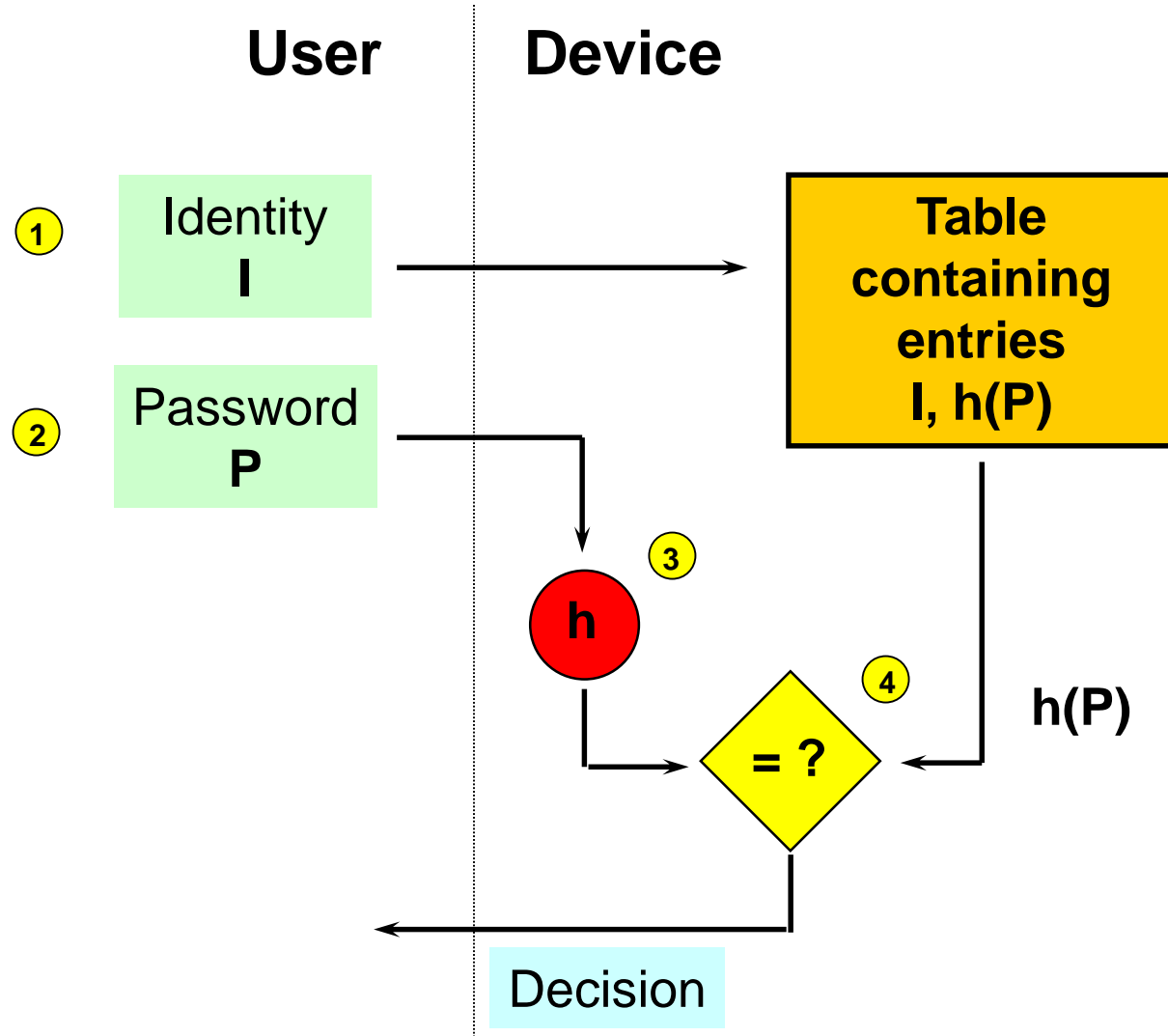
*“Now, a few words on looking for things. When you go looking for something specific, your chances of finding it are very bad. Because of all the things in the world, you're only looking for one of them. When you go looking for anything at all, your chances of finding it are very good. Because of all the things in the world, you're sure to find some of them.”*

*Daryl Zero, in Zero Effect*





# Application requiring preimage resistance



# XKCD's take on this...

HACKERS RECENTLY LEAKED **153 MILLION** ADOBE USER EMAILS, ENCRYPTED PASSWORDS, AND PASSWORD HINTS.

ADOBE ENCRYPTED THE PASSWORDS IMPROPERLY, MISUSING BLOCK-MODE 3DES. THE RESULT IS SOMETHING WONDERFUL:

USER	PASSWORD	HINT	
4e18acc1ab27b2d6	4e18acc1ab27b2d6	WEATHER VANE SWORD	<input type="text"/>
4e18acc1ab27b2d6	4e18acc1ab27b2d6	NAME 1	<input type="text"/>
8bab66279e06eb6d	8bab66279e06eb6d	DUH	<input type="text"/>
8bab66279e06eb6d	8bab66279e06eb6d	57	<input type="text"/>
8bab66279e06eb6d	8bab66279e06eb6d	FAVORITE OF 12 APOSTLES	<input type="text"/>
4e18acc1ab27b2d6	4e18acc1ab27b2d6	WITH YOUR OWN HAND YOU	<input type="text"/>
1ab29ae86da6e5ca	1ab29ae86da6e5ca	HAVE DONE ALL THIS	<input type="text"/>
a1f9b2b6299e7a2b	a1f9b2b6299e7a2b	SEXY EARLOBES	<input type="text"/>
a1f9b2b6299e7a2b	a1f9b2b6299e7a2b	BEST TOS EPISODE	<input type="text"/>
39738b7adb0b8af7	39738b7adb0b8af7	SUGARLAND	<input type="text"/>
1ab29ae86da6e5ca	1ab29ae86da6e5ca	NAME + JERSEY #	<input type="text"/>
877ab7689d3862b1	877ab7689d3862b1	ALPHA	<input type="text"/>
877ab7689d3862b1	877ab7689d3862b1		<input type="text"/>
877ab7689d3862b1	877ab7689d3862b1		<input type="text"/>
877ab7689d3862b1	877ab7689d3862b1	OBVIOUS	<input type="text"/>
877ab7689d3862b1	877ab7689d3862b1	MICHAEL JACKSON	<input type="text"/>
38a7c9279cdeb44	38a7c9279cdeb44	HE DID THE MASH, HE DID THE	<input type="text"/>
38a7c9279cdeb44	38a7c9279cdeb44	PURLOINED	<input type="text"/>
38a7c9279cdeb44	38a7c9279cdeb44	FAV. LATER-3 POKEMON	<input type="text"/>

THE GREATEST CROSSWORD PUZZLE  
IN THE HISTORY OF THE WORLD

# Application requiring 2nd preimage resist.

---

Hash functions are commonly used to generate checksums to provide a level of data integrity.

## **Choose Download Location**

### **mediaCam AV 2.7**

You have chosen to download **mediaCam AV 2.7**. Check the file details to make sure this is the correct program and version, and that your operating system is supported.

### **Download Details**

Operating Systems 98/2k/Me/XP

File Name mediaCamAV2.7.2.0\_Installer.exe

MD5 Hash 8642009dfd6658e0399586fb27134886

File Size 4.27 MB (4,474,880 bytes)

# Hash functions and data integrity

---

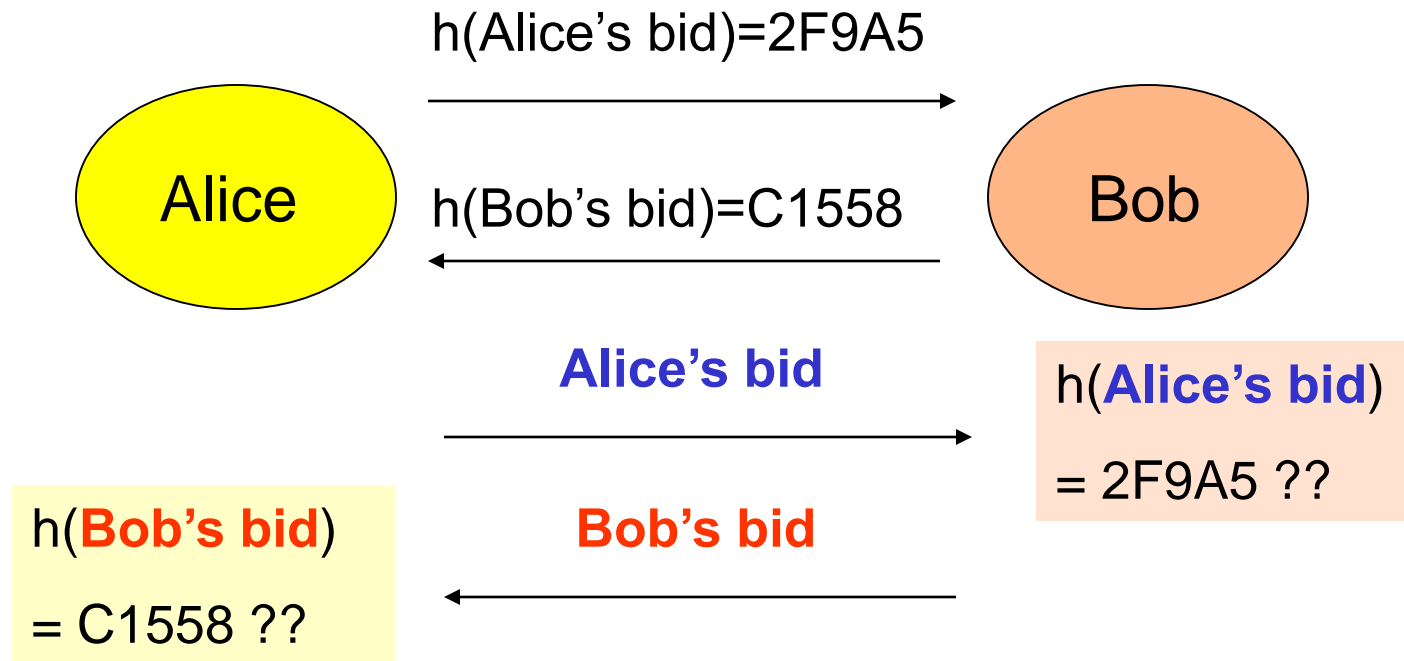


**To what extent can a hash function provide a data integrity service?**

# Application requiring collision resistance

Suppose Alice and Bob want to openly bid for a contract amongst themselves (lowest bid wins):

Who bids first?



# Hash function collisions

---

Because a hash is shorter than the message, collisions are inevitable – we just want them to be hard to find.

**How long does a hash have to be before finding collisions is hard?**

# Hash function collisions

---

Suppose that we hash the message **Keith owes Fred £10** using a hash function that has a hash of just 2 bits:

there are only four possible hashes: 00, 01, 10 or 11.

Fred receives the hashed message, and being a manipulative type he decides to argue that it corresponds to the message **Keith owes Fred £100**.



What is the probability that:

**hash (Keith owes Fred £10 ) = hash (Keith owes Fred £100 )?**

# Hash function collisions

---

Suppose the hash is 10 bits long – in other words about 1000 hashes

## 1000 requests for £200

1. Pay Fred Piper £200
2. Pay F. Piper £200
3. Pay F.C. Piper two hundred pounds
4. Pay F.C. Piper two hundred pounds only
5. Pay two hundred pounds to Mr Fred Piper
6. ....

## 1000 request for £8000

1. Pay Fred Piper £8000
2. Pay F. Piper £8000
3. Pay F.C. Piper eight thousand pounds
4. Pay F.C. Piper eight thousand pounds only
5. Pay eight thousand pounds to Mr Fred Piper
6. ....

Since there are only 1000 different possible values of the hash, there is a **very good chance** that there will be at least one match...

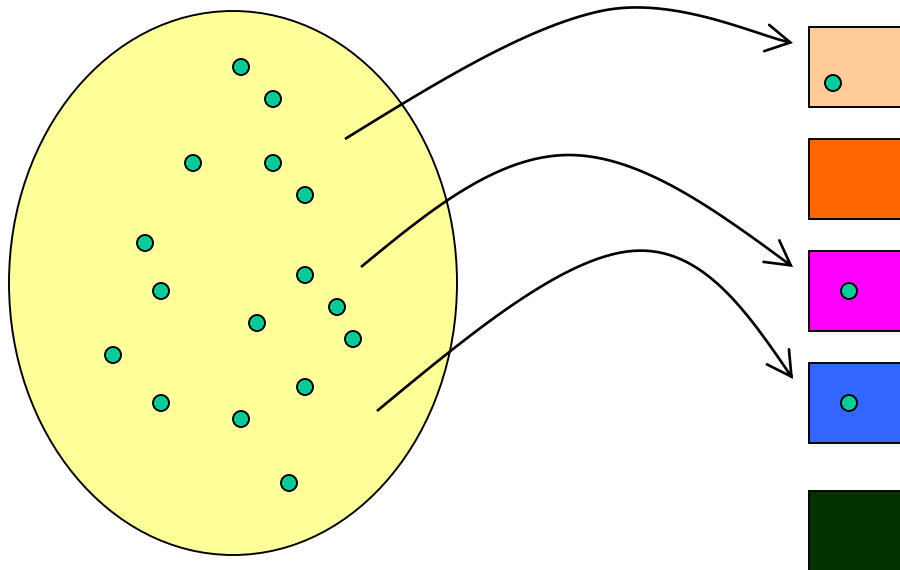


# Birthday attacks

---

Consider an experiment where we take  $Q$  balls and start throwing them into  $M$  bins (where  $M$  is a smaller number than  $Q$ ).

**After how many throws there is a greater than half chance that one bin contains two balls?**



# Birthday attacks

---

If we take  $Q$  balls and start throwing them into  $M$  bins then **after the square root of  $M$  throws** there is a greater than half chance that one bin contains two balls.

If we take  $Q$  messages and start hashing them using a hash function whose output is  $r$  bits then **after the square root of  $2^r$  hashes** there is a greater than half chance that two messages have the same hash value (in other words, we have a collision).

$$\text{square root of } 2^r = 2^{r/2}$$

# Examples of hash functions

---

- **(MD4) MD5**
  - 128 bit hash
  - RFC1321, used for file integrity checking
- **(SHA-0) SHA-1**
  - 160-bit hash
  - Used in TLS/SSL, PGP, SSH, S/MIME, IPsec
- **SHA-2 (SHA224, SHA256, SHA384, SHA512)**
  - Recommended for U.S. government use
- **(RIPEMD) RIPEMD-160**
- **Whirlpool**
  - 512-bit hash (based on modified AES)
  - Used in TrueCrypt (open source encryption toolkit)

# Hash function cryptanalysis

---

1996	Collisions for MD4 found with complexity $2^{20}$
Aug04	Collisions for MD5 found with complexity $2^{39}$
Aug04	Collisions found for SHA-0
2005	Preimage attacks against MD4
2005	Collisions for SHA-1 found with complexity $2^{63}$
2012	Collisions for SHA-1 estimated at complexity $2^{61}$
2017	Leading browsers to stop supporting SHA-1 in SSL certificates

# SHA-3

---

- In 2007, NIST launched an AES-style competition to design new hash functions.
- The closure of the call period was the end of 2008.
- 63 submissions received, 51 selected for first round
- Two conferences to discuss candidates held
- 14 semi-finalists analysed
- 5 finalists selected at end of 2010
- Keccak announced as winner in 2012
- NIST SHA-3 standard (FIPS 202) published Aug 2015

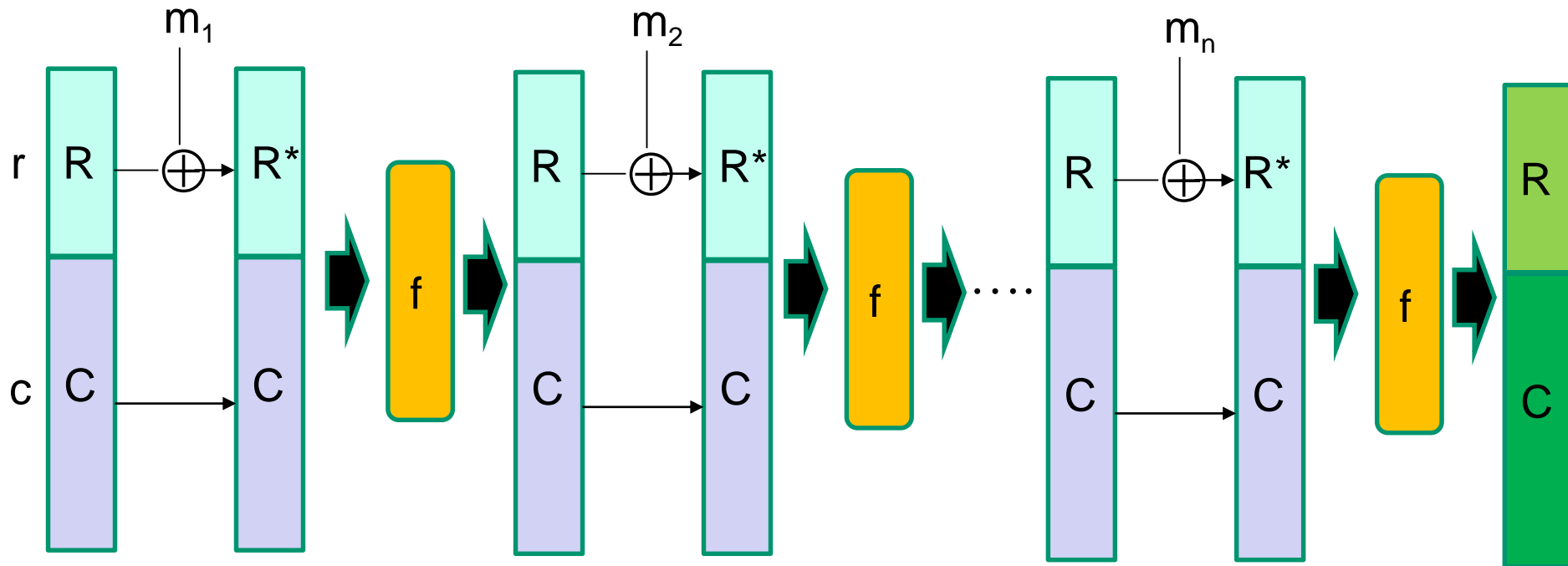
# SHA-3

---

- Keccak is designed by industry-based researchers from STMicroelectronics and NXP, including Joan Daemen (AES co-designer)
- Based on a new design approach (a “sponge” construction)
- Variable length output
- Variable throughput, allowing efficiency/security trade-offs
- Chosen for elegance, security margins, performance and flexibility
- Intended to complement, not replace, SHA-2

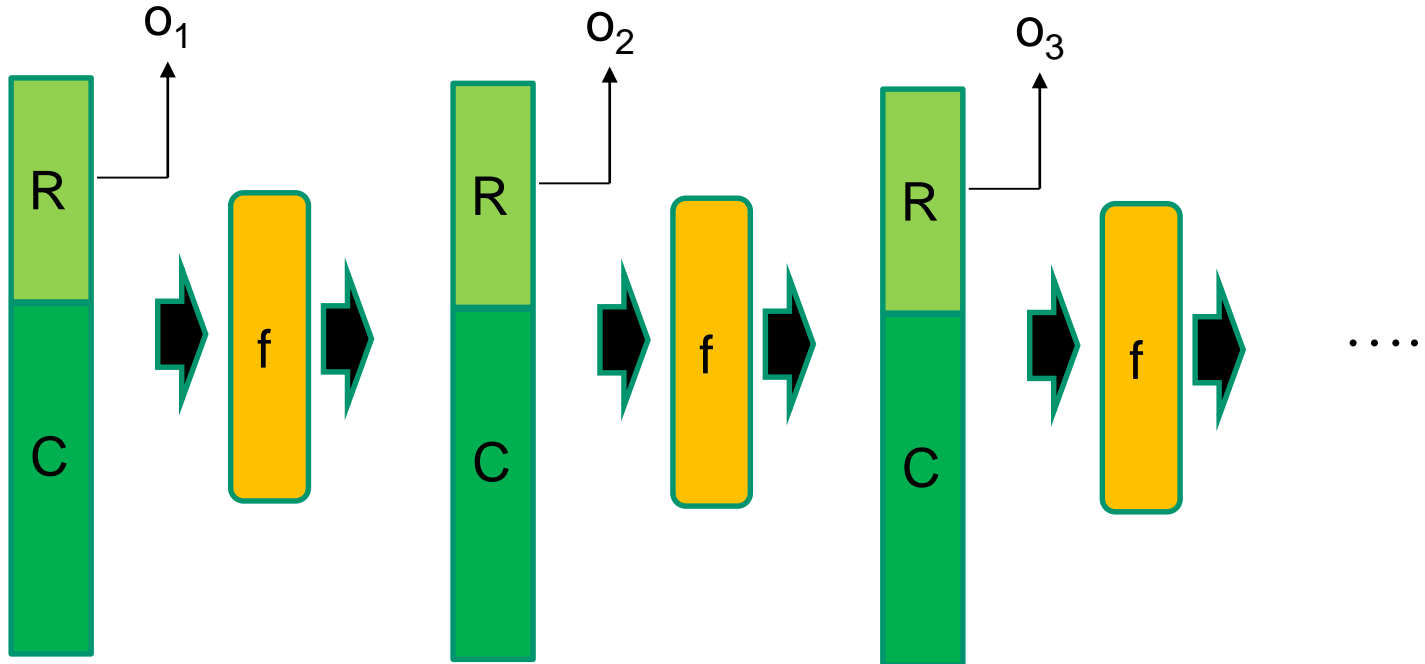
# Sponge construction input

$$m = m_1 m_2 \dots m_n$$



# Sponge construction output

---



$$h(m) = o_1 o_2 o_3 \dots$$



---

## 2. Message Authentication Codes

# Active attacks

---

Recall the following list of active attacks:

- Changing part of a message
- Deletion of part of a message
- Insertion of a false message
- Falsifying the origin of a message



**Does encryption prevent these attacks?**

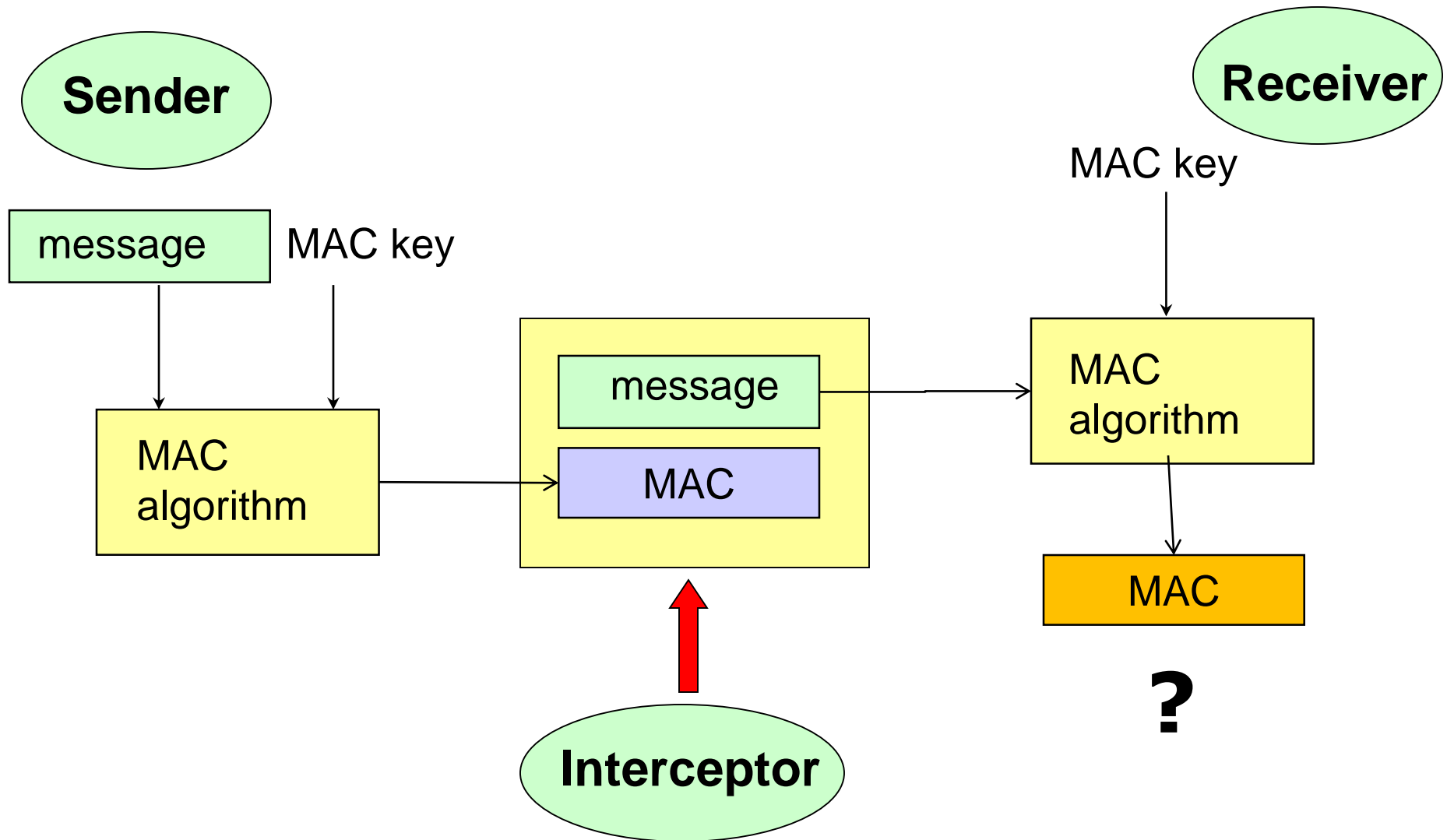
# Strong data integrity mechanisms

---

We will consider two different types of data integrity mechanism that also provide data origin authentication:

1. **Message Authentication Codes (MACs)** are symmetric mechanisms that we consider now.
2. **Digital signatures** are asymmetric mechanisms that we consider later.

# Basic model of a MAC



# MAC properties

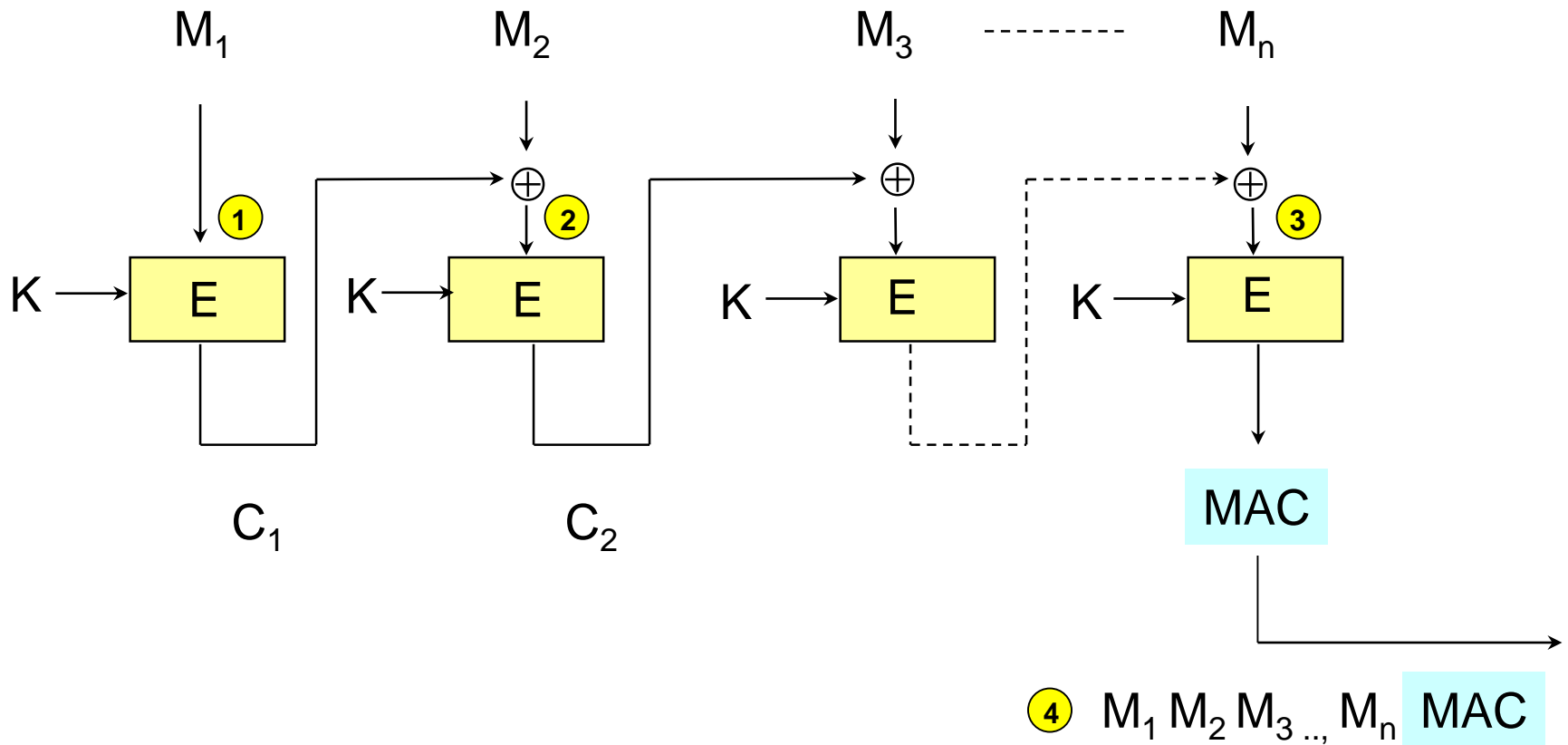
---

- Condenses arbitrarily long inputs into a fixed length output
- Easy to compute for anyone who knows the MAC key
- It is hard to find a valid message / MAC pair without knowledge of the MAC key

MACs are typically constructed from block ciphers or hash functions

# CBC-MAC

(Padded) message divided into blocks



# MAC question

---



What are the pros and cons of just sending part of the last ciphertext block as the MAC?

# “Textbook cryptography” alert!

---

CBC-MAC as described on the earlier slide is insecure!

Secure standards for variants of CBC-MAC specify additional processing that must be done after the last encryption, before creating the MAC



# HMAC

---

RFC 2104 describes how to convert a hash function into a MAC

- Let K1 and K2 be two cryptographic keys
- Let h be a hash function

$$\text{HMAC}(\text{message}) = h( K1 \parallel h( K2 \parallel \text{message} ) )$$



What is the length of an HMAC key?

---

## 3. Authenticated encryption

# MAC then encrypt

---

- Compute the MAC on the message
- Encrypt the message (and the MAC)
- Send the ciphertext (and the MAC)



What are the disadvantages of this approach?

# Encrypt then MAC

---

- Encrypt the message
- Compute the MAC on the ciphertext
- Send the ciphertext and the MAC



What are the disadvantages of this approach?

# Authenticated-encryption primitives

---

- An **authenticated-encryption primitive** uses a single symmetric key to provide both confidentiality and data origin authentication
- Increasingly adopted by a wide range of applications
- Examples of a-e primitives include:
  - Counter with CBC-MAC (CCM) mode
  - EAX mode
  - Offset Codebook (OCB) mode
  - Galois Counter (GCM) mode

# Summary

---

- Hash functions are important versatile cryptographic primitives that are widely employed
- Message authentication codes are the most important mechanism for providing data origin authentication using symmetric cryptography
- Authenticated-encryption primitives provide efficient methods of offering both confidentiality and data origin authentication