# ENEE 457: Computer Systems Security
# 09/19/16

# Lecture 6
# Message Authentication Codes and Hash Functions

**Charalampos (Babis) Papamanthou**

Department of Electrical and Computer Engineering
University of Maryland, College Park

- Slides adjusted from:
  - http://dziembowski.net/Teaching/BISS09/

# Some practictioners don’t like the CBC-MAC

We **don’t want** to authenticate using the **block ciphers**!

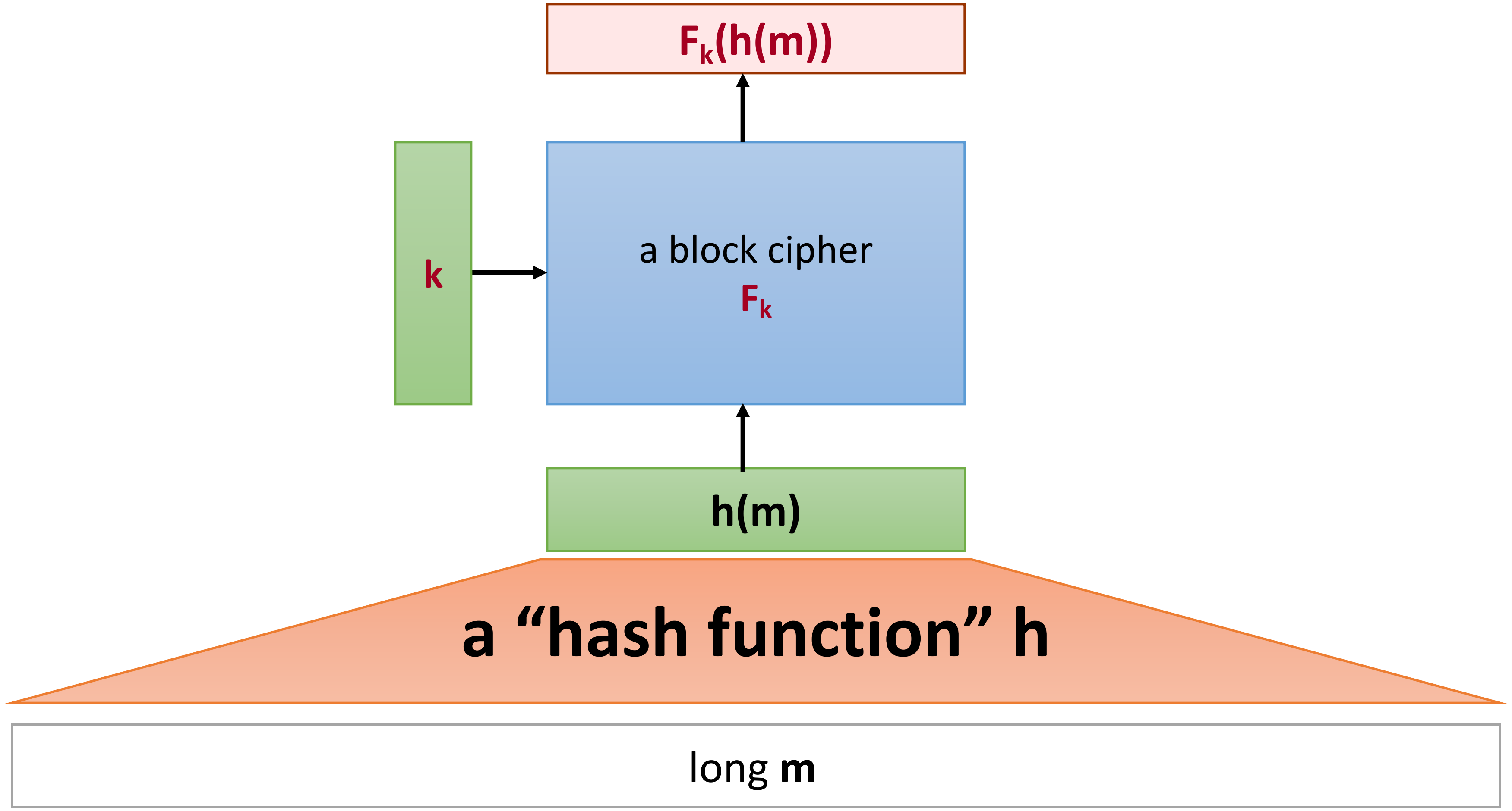
What do you want to use instead?

Hash functions!

Why?

**Because** they are **more efficient**

# Another idea for authenticating long messages



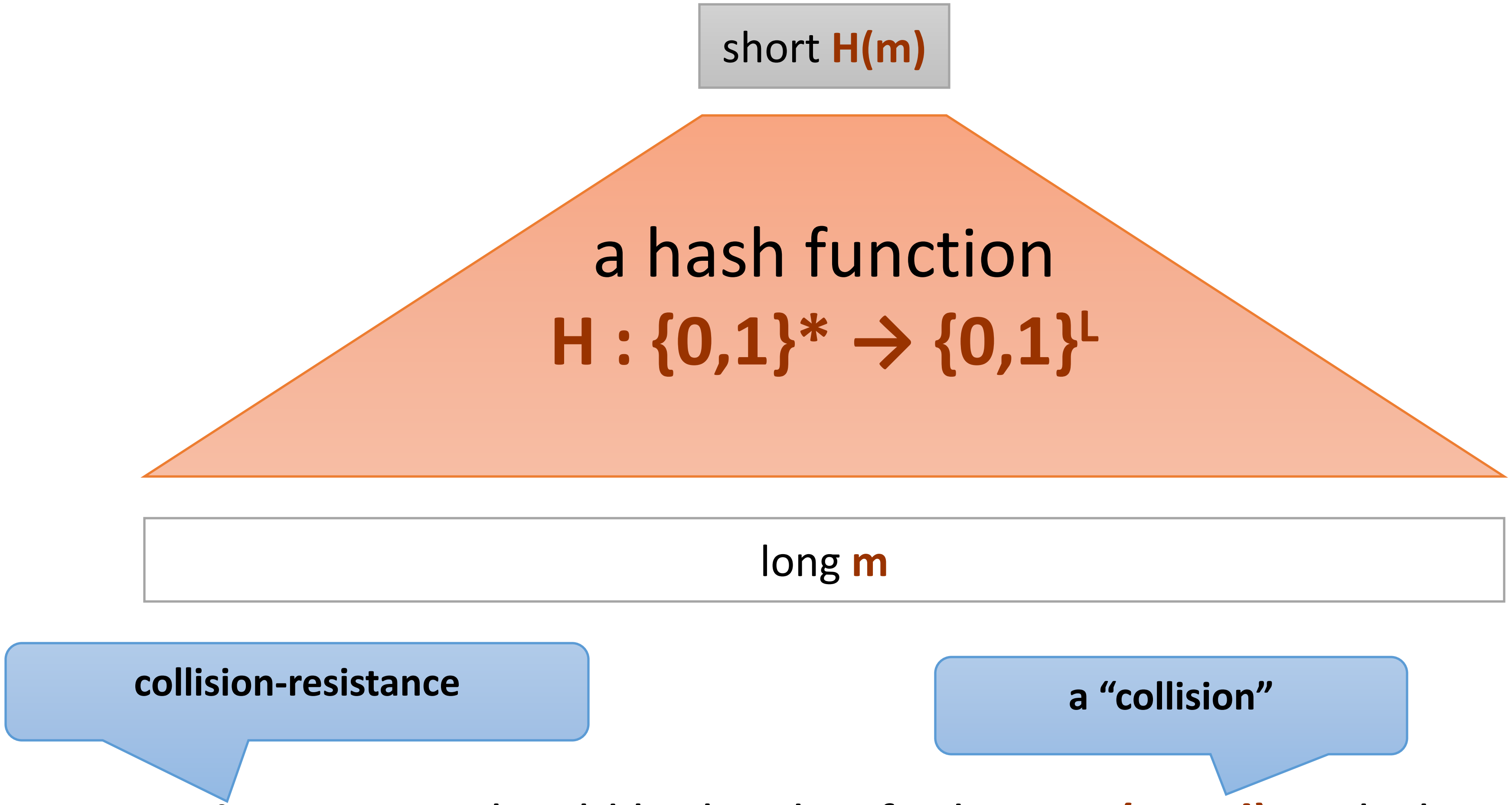
# How to formalize it?

We need to define what is a “hash function”.

The basic property that we require is:

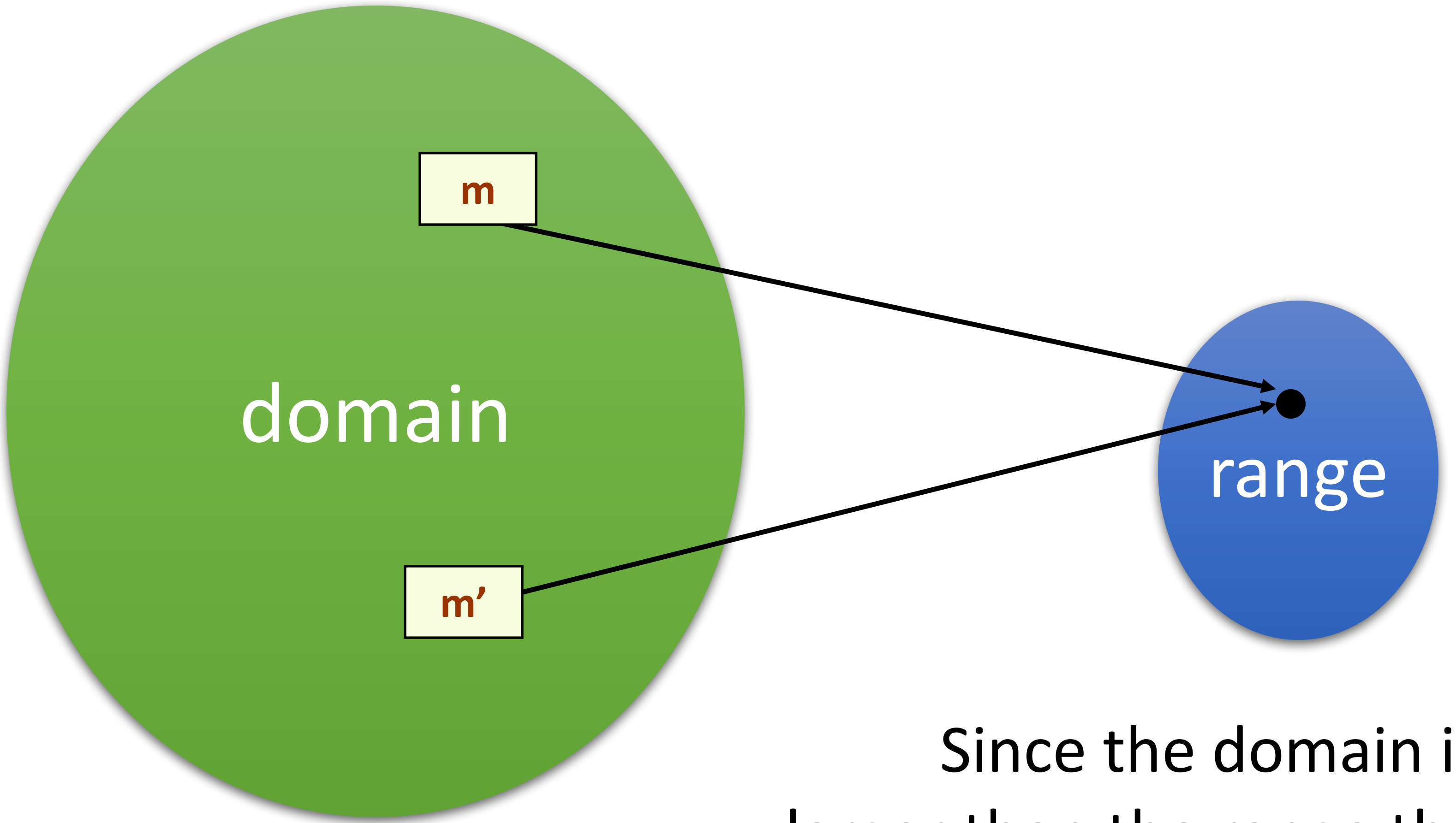
“collision resistance”

# Collision-resistant hash functions




**Requirement**: it should be hard to find a pair **(m,m')** such that **H(m) =H(m')**

# Collisions always exist




Since the domain is larger than the range the collisions have to exist.

# “Practical definition”

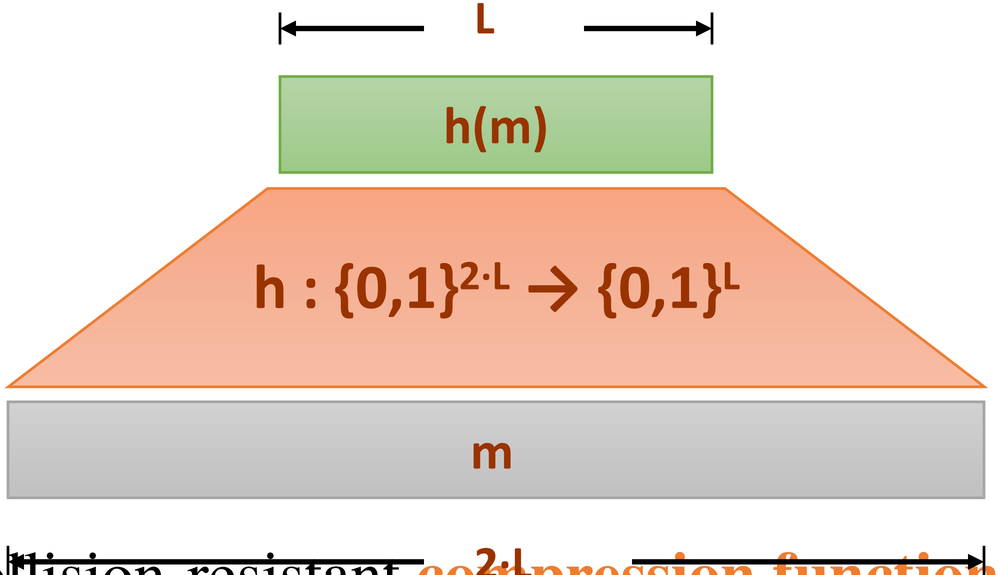
**H** is a **collision-resistant hash function** if it is “*practically impossible to find collisions in **H***”.

**Popular hash funcitons:**

- **MD5** (now considered broken)
- **SHA1**
- **...**

# A common method for constructing hash functions

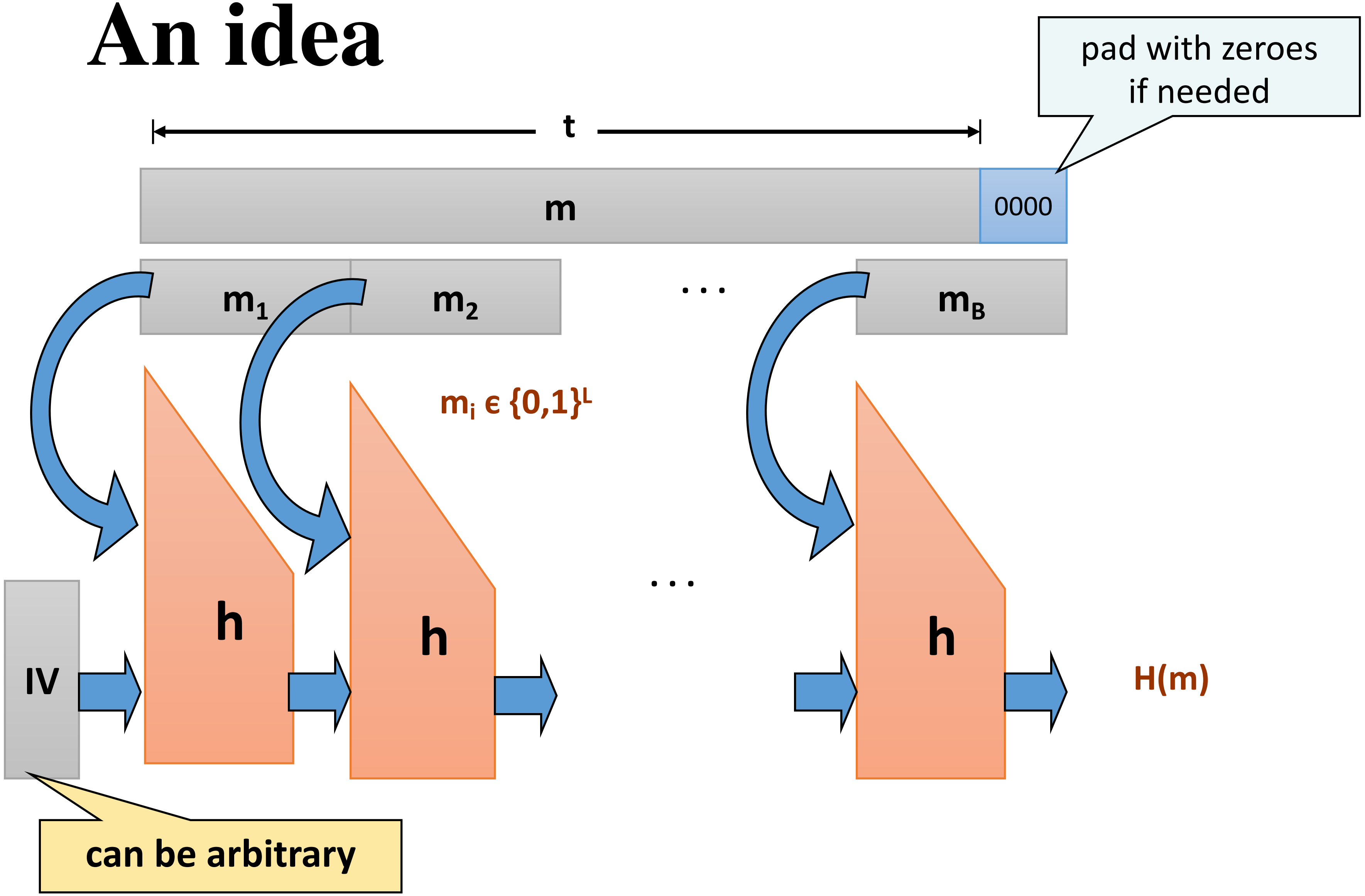
1. Construct a "***fixed-input-length***" collision-resistant hash function




Call it: a collision-resistant **compression function**.

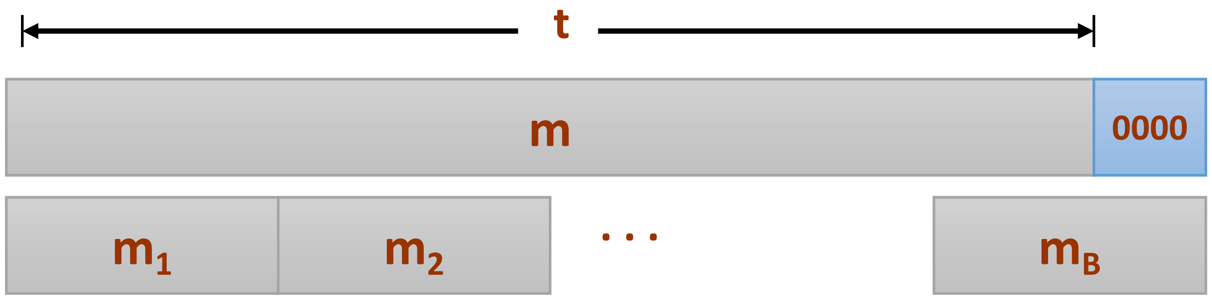
2. Use it to construct a hash function.

# An idea




This doesn't work...

# Why is it wrong?

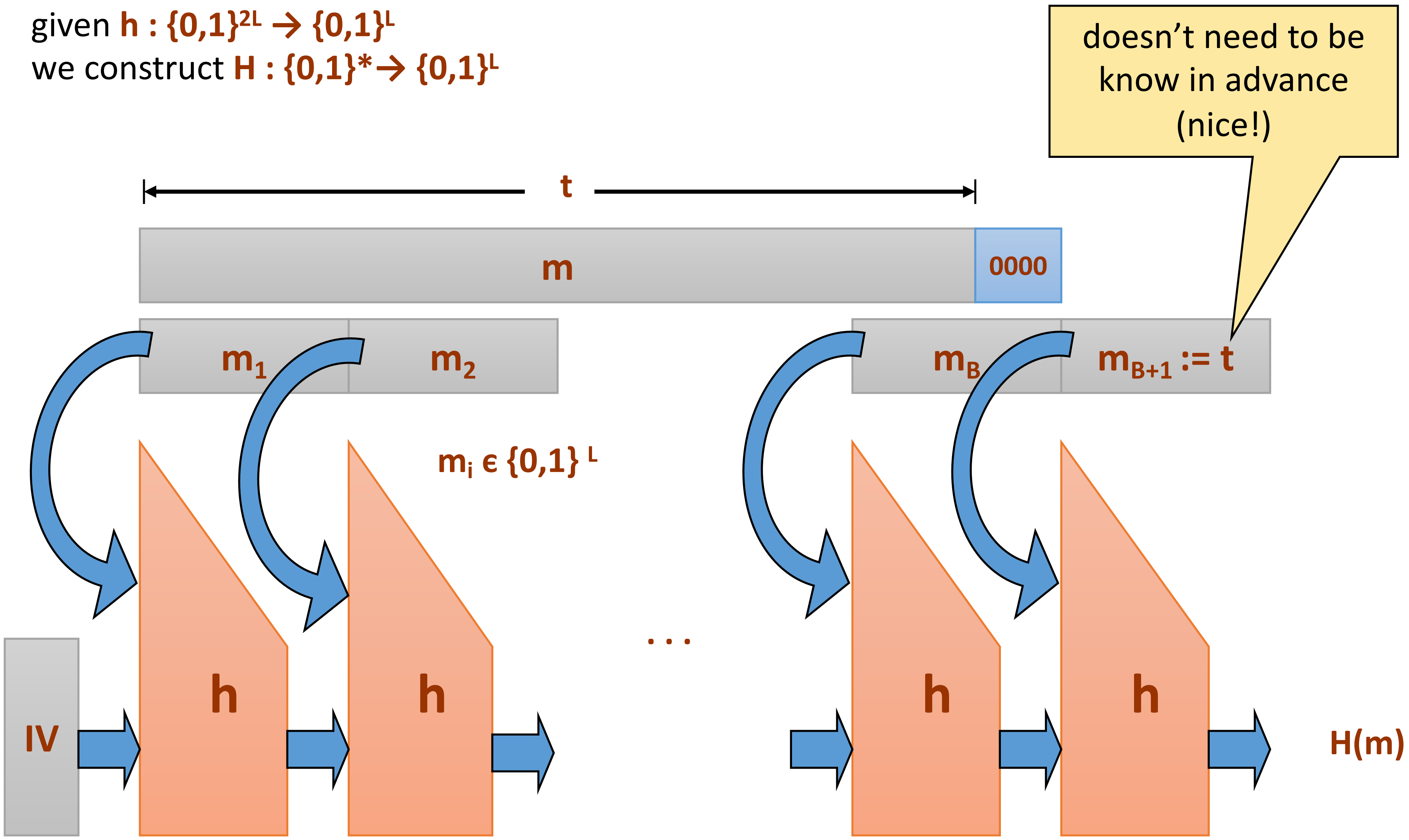



If we set **m' = m || 0000** then **H(m') = H(m).**

**Solution**: add a block encoding "**t**".

# Merkle-Damgård transform

given **h : $\{0,1\}^{2L} \rightarrow \{0,1\}^L$**
we construct **H : $\{0,1\}^* \rightarrow \{0,1\}^L$**



# This construction is secure

We would like to prove the following:

**Theorem**

If

$$h : \{0,1\}^{2L} \rightarrow \{0,1\}^{L}$$

is a collision-resistant **compression** function

then

$$H : \{0,1\}^{*} \rightarrow \{0,1\}^{L}$$

is a collision-resistant **hash** function.

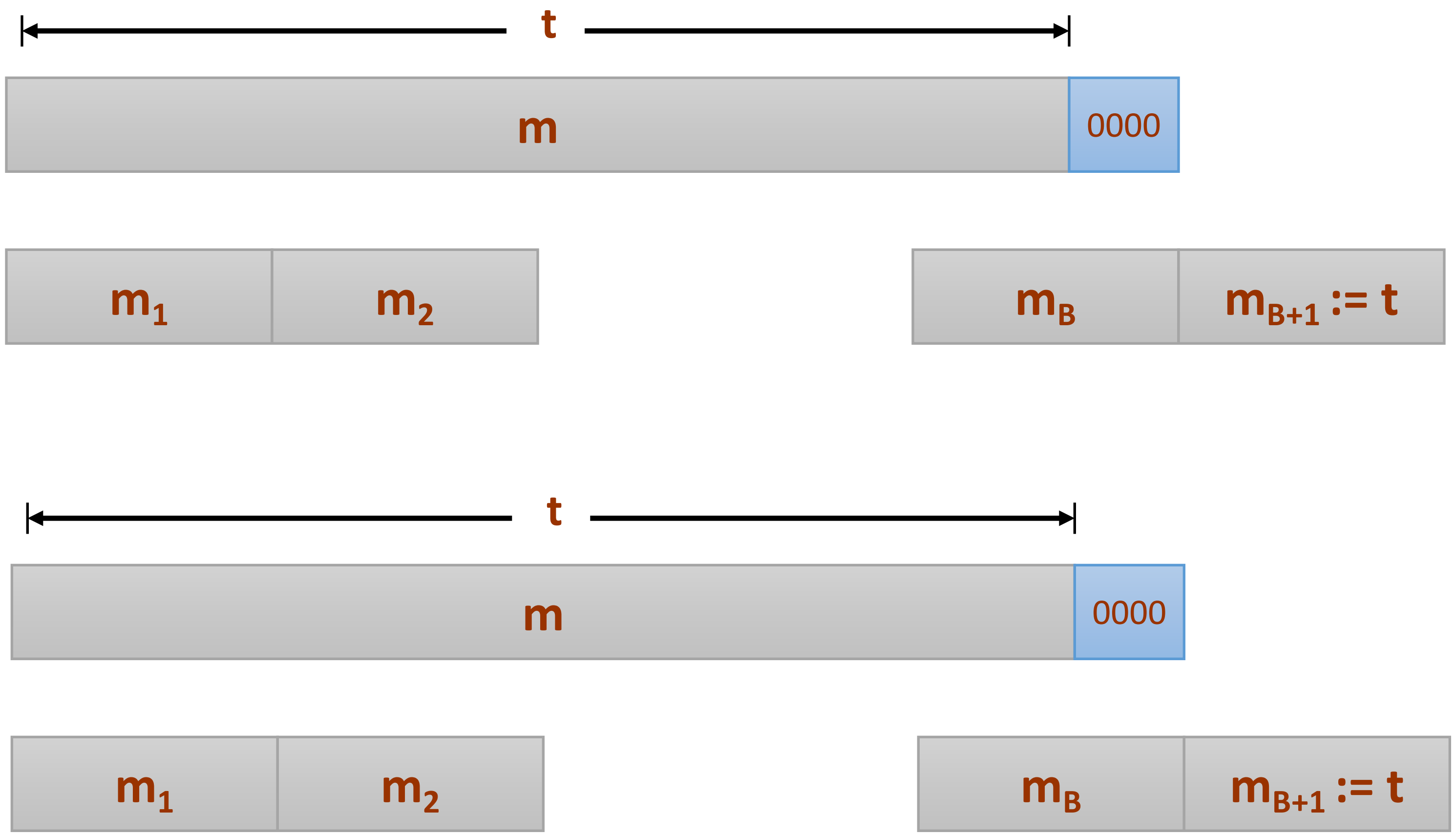
# Let’s prove it: How to compute a collision (x,y) in h from a collision (m,m’) in H?

We consider two options:

1. |m| = |m’|

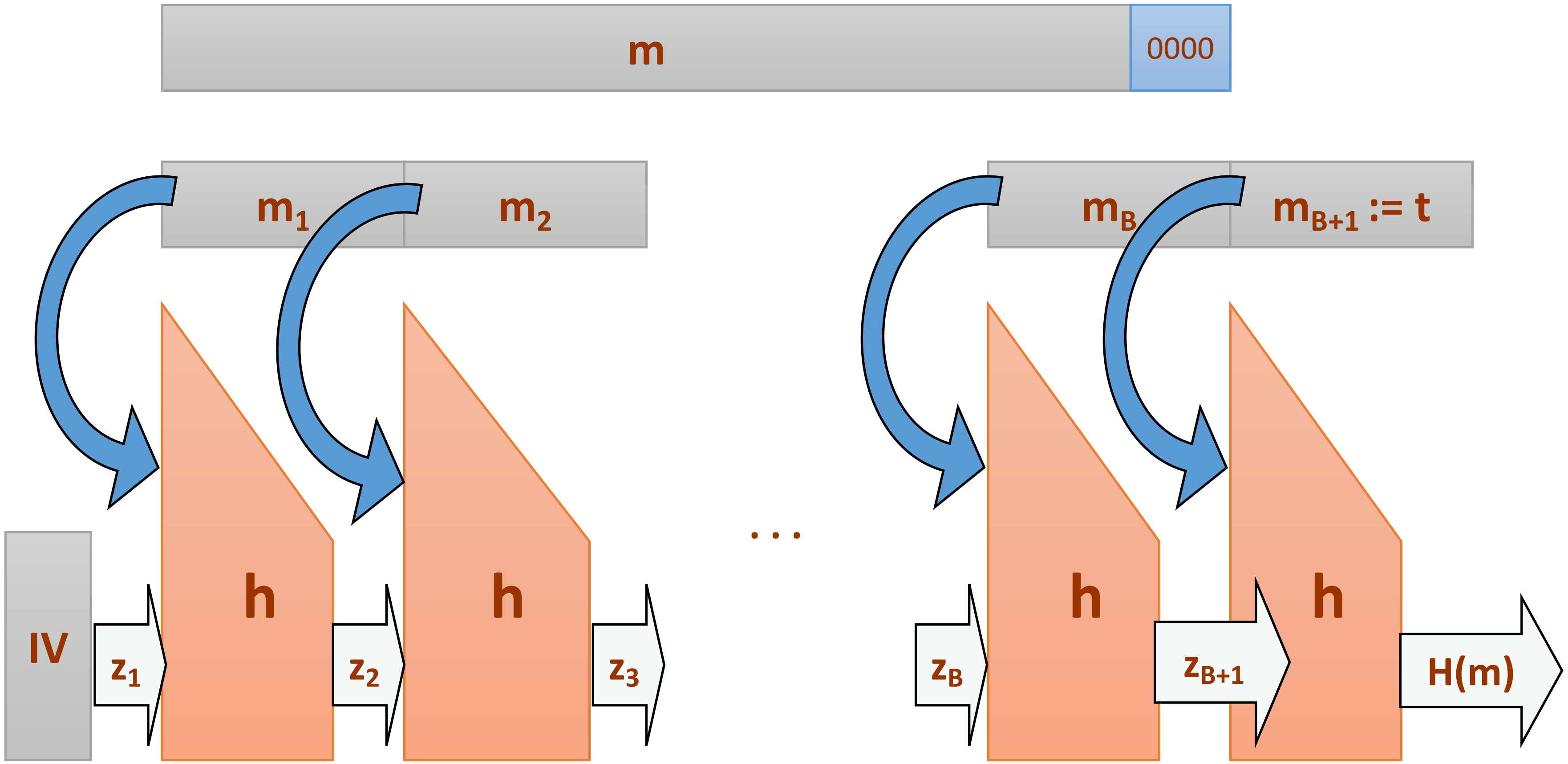
2. |m| ≠ |m’|

# Option 1: |m| = |m'|



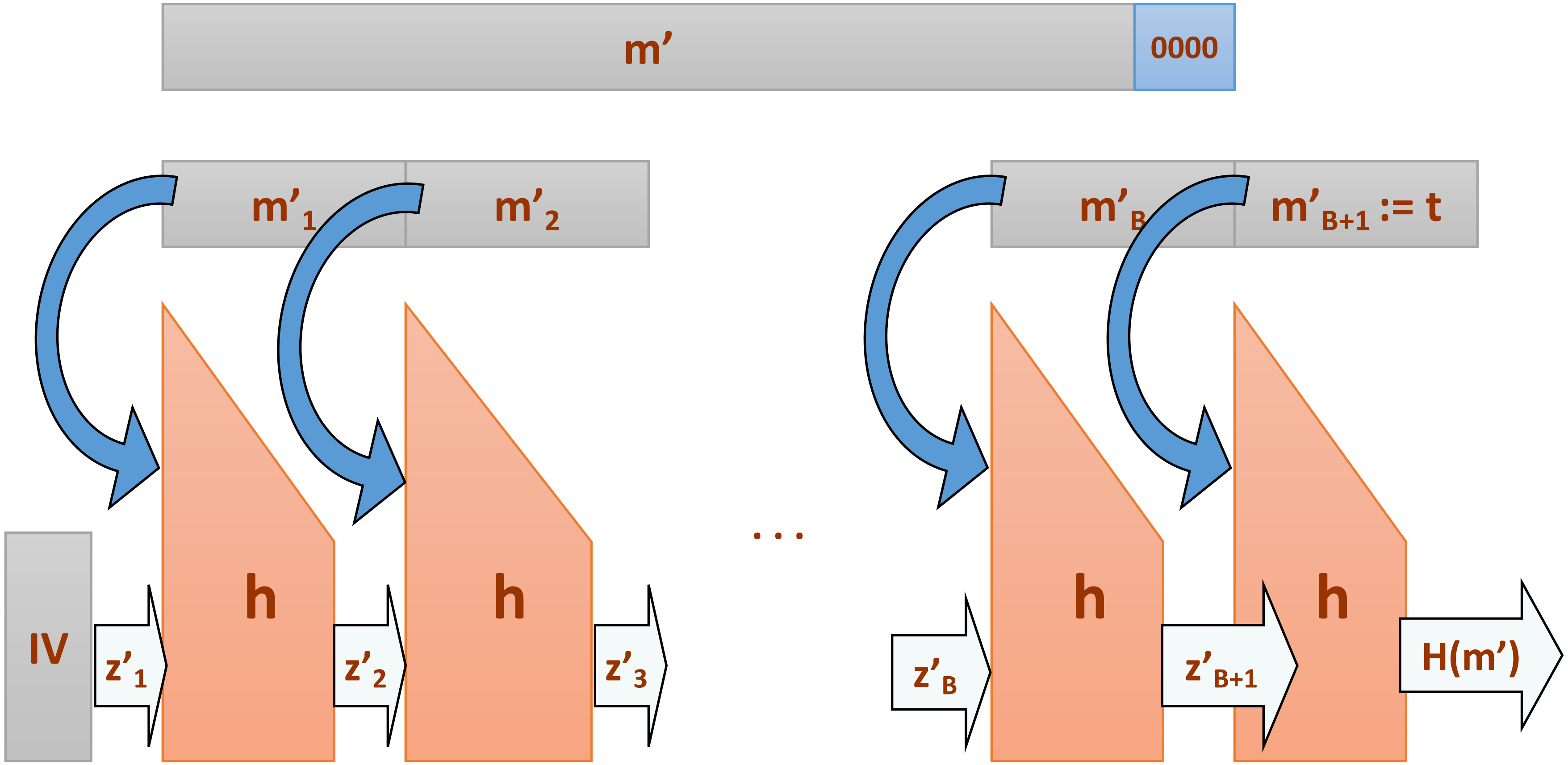
# |m| = |m'|

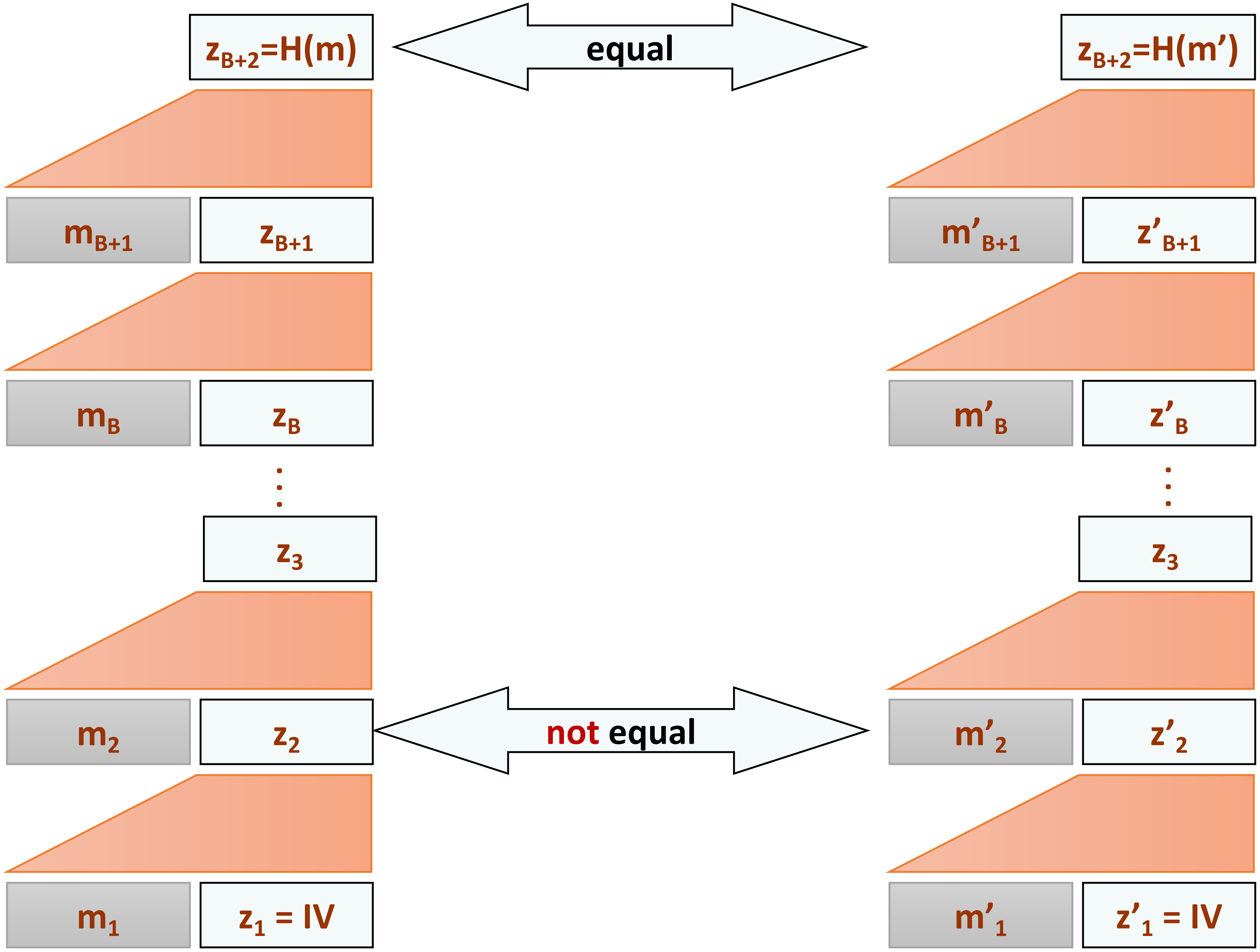
Some notation:

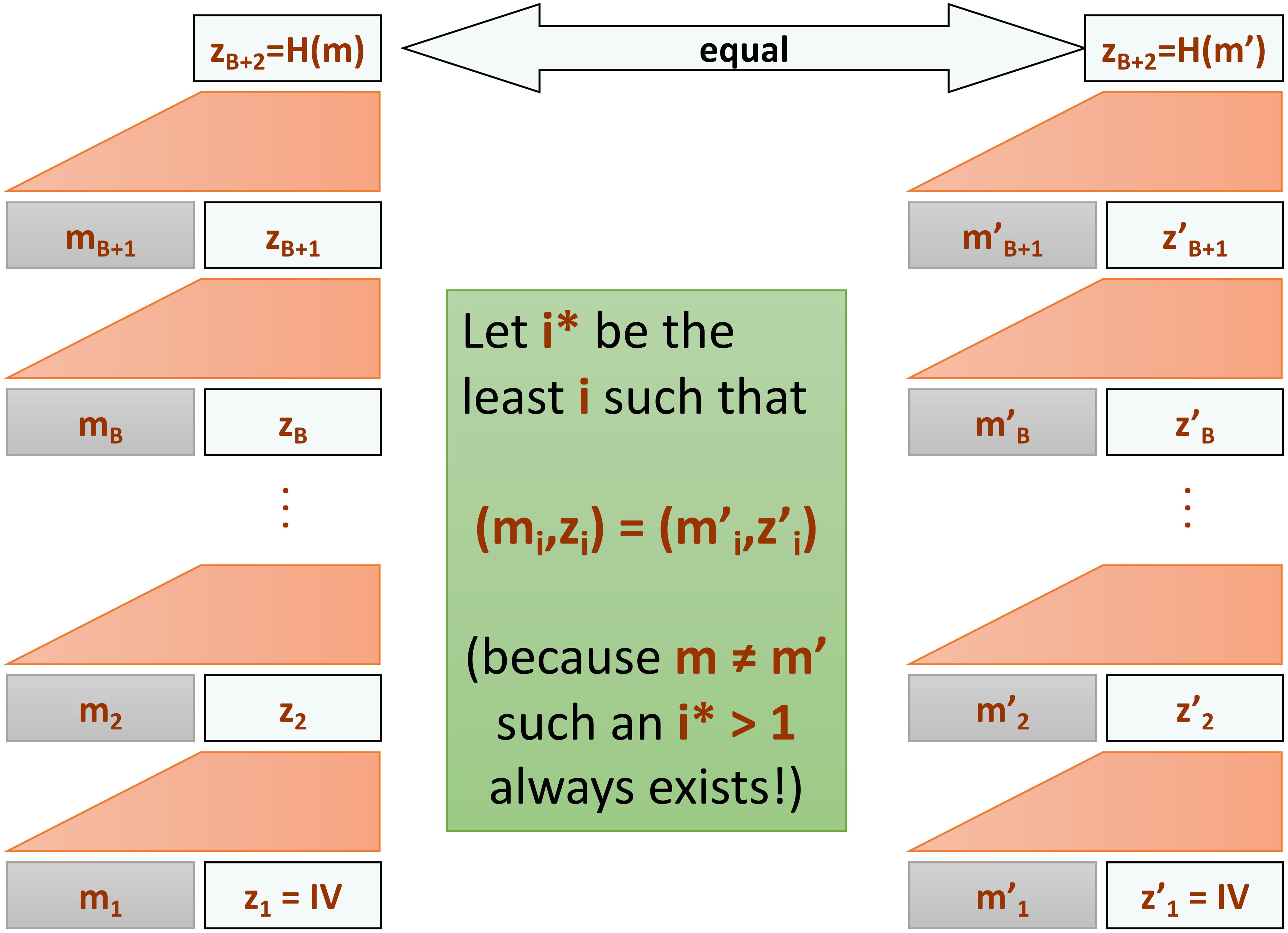


# |m| = |m'|

For **m'**:




z_{B+2}=H(m)
equal
z_{B+2}=H(m')
m_{B+1}
z_{B+1}
m'_{B+1}
z'_{B+1}
m_B
z_B
m'_B
z'_B
z_3
z_3
m_2
z_2
not equal
m'_2
z'_2
m_1
z_1 = IV
m'_1
z'_1 = IV

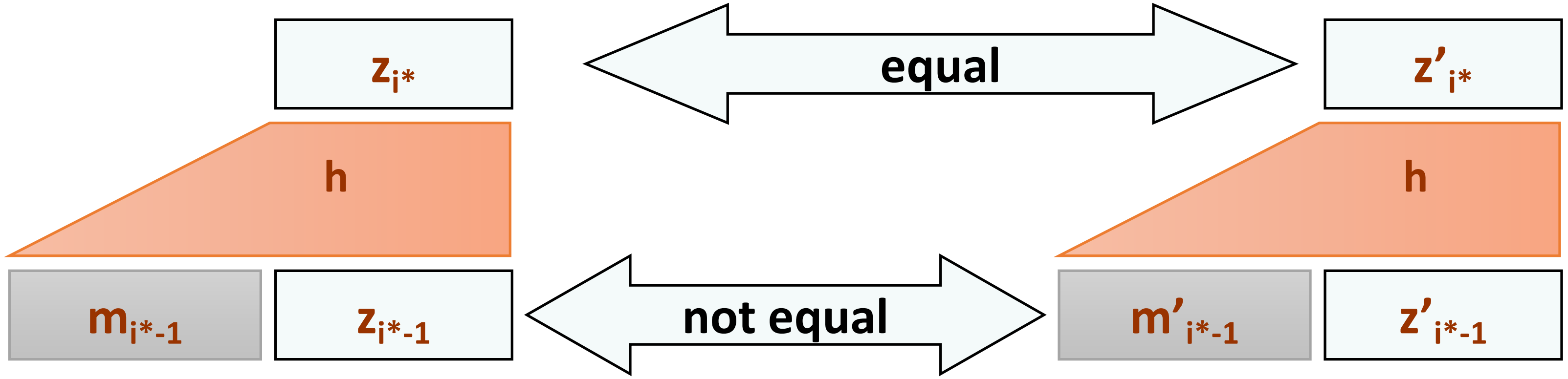



Let **i*** be the least **i** such that

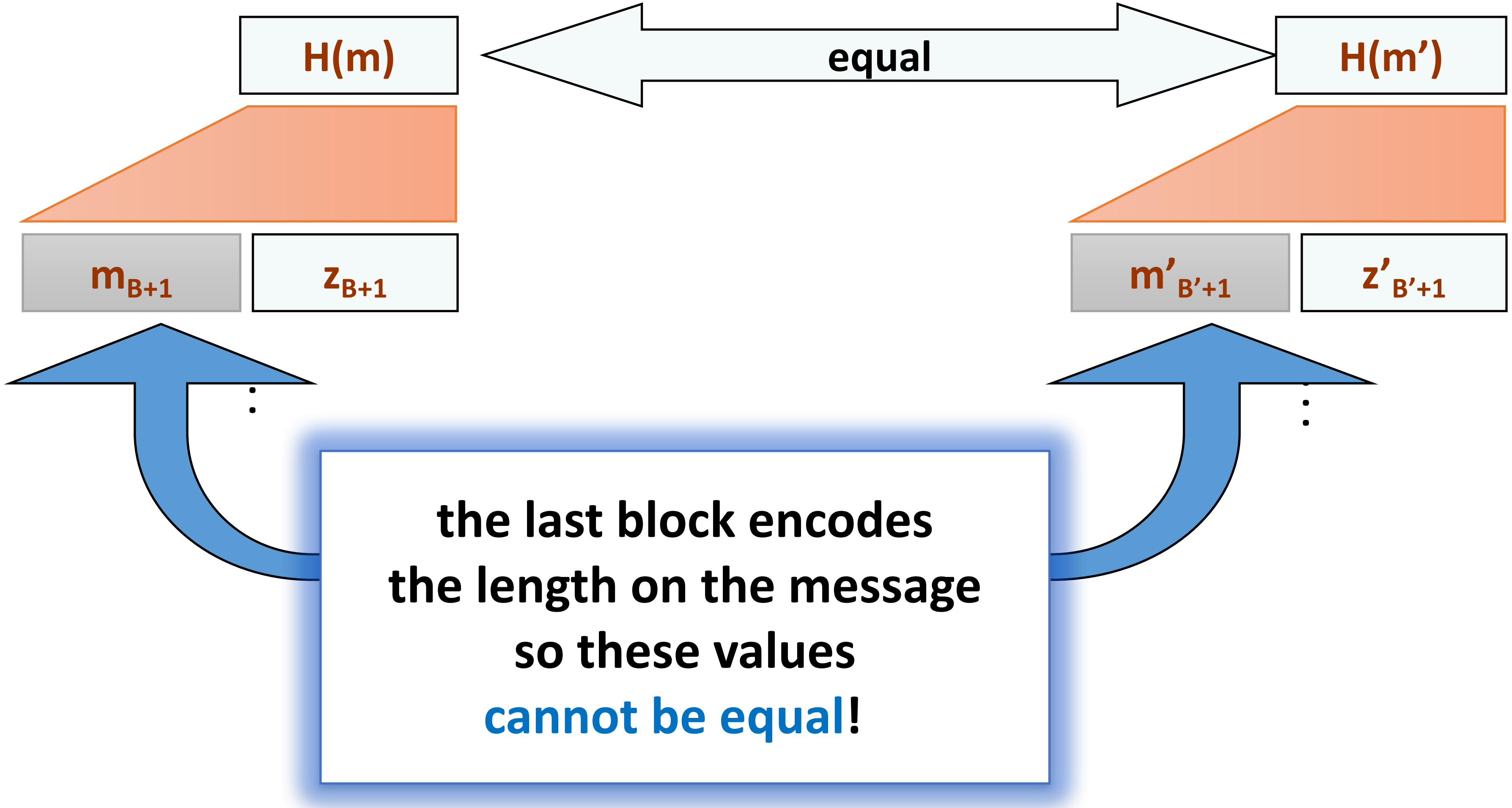
**$(m_i,z_i) = (m'_i,z'_i)$**

(because **m ≠ m'** such an **i* > 1** always exists!)

# So, we have found a collision!



# Option 2: |m| ≠ |m'|




So, again we have found a collision!

# Generic attacks on hash functions

Remember the **brute-force** attacks on the encryption schemes?

For the **hash functions** we can do something **slightly smarter**...

It is called a "**birthday attack**".

# The birthday paradox

Suppose we have a random function

$$\mathbf{H : A \rightarrow B}$$

Take **n** values

$$\mathbf{x_1,\dots,x_n}$$

Let **p(n)** be the probability that there exist distinct **i,j** such that

$$\mathbf{H(x_i) = H(x_j).}$$

If **n ≥ |B|** then trivially **p(n) = 1**.

**Question:** How large **n** needs to be to get **p(n) = 1/2**

**Answer:** $n \approx \sqrt{|B|}$

# Why is it called "a birthday paradox"?

Set:

**H : people → birthdays**

**Q:** How many random people you need to take to know that with probability **0.5** at least **2** of them have birthday on the same day?

**A:** **23** is enough!

**Counterintuitive...**

# How does the birthday attack work?

For a hash function

$$H : \{0,1\}^* \rightarrow \{0,1\}^L$$

Take a random **X** – a subset of $\{0,1\}^{2L}$, such that $|X| = 2^{L/2}$.

With probability around **0.5** there exists $x,x' \in X$, such that

$$H(x) = H(x').$$

A pair **(x,x')** can be found in time $O(|X| \log |X|)$ and space $O(|X|)$.

**Moral**

**L** has to be such that an attack that needs $2^{L/2}$ steps is infeasible.

# Find collisions for crypto-hashes?

- The brute-force birthday attack aims at finding a collision for a cryptographic function h with domain [1,2,…,m]
  - Randomly generate a sequence of plaintexts $X_1, X_2, X_3,\ldots$
  - For each $X_i$ compute $y_i = h(X_i)$ and test whether $y_i = y_j$ for some $j < i$
  - Stop as soon as a collision has been found
- If there are m possible hash values, the probability that the i-th plaintext does not collide with any of the previous $i - 1$ plaintexts is $1 - (i - 1)/m$
- The probability $F_k$ that the attack fails (no collisions) after k plaintexts is

$$F_k = (1 - 1/m)\,(1 - 2/m)\,(1 - 3/m)\,\ldots\,(1 - (k - 1)/m)$$

- Using the standard approximation $1 - x \approx e^{-x}$

$$F_k \approx e^{-(1/m + 2/m + 3/m + \ldots + (k-1)/m)} = e^{-k(k-1)/2m}$$

- The attack succeeds with probability p when $F_k = 1 - p$, that is,

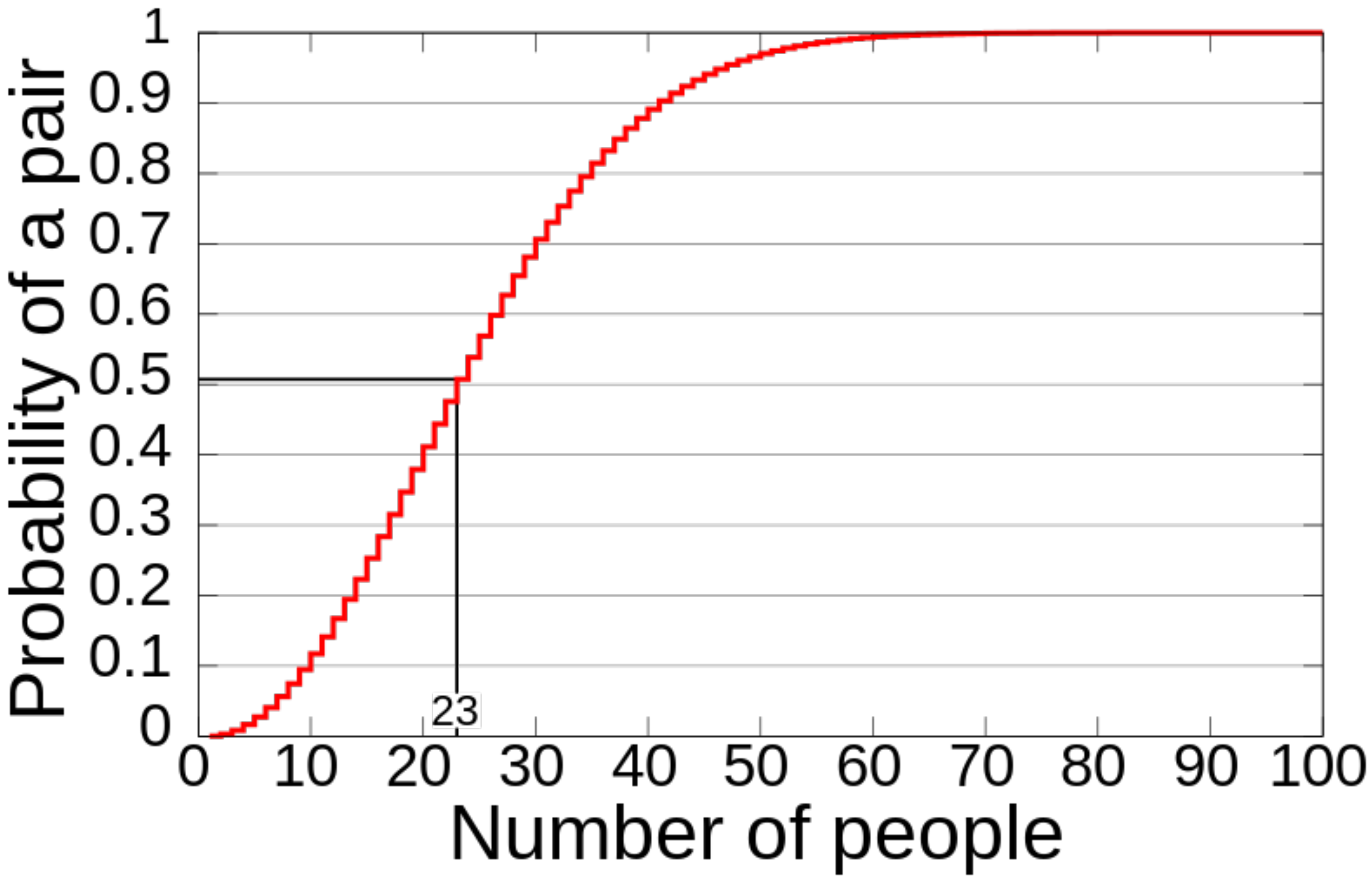
$$e^{-k(k-1)/2m} = 1 - p$$

- For p=1/2

$$k \approx 1.17\, m^{½}$$

- For m = 365, p=1/2, k is around 24

# Birthday attack



# Concrete functions

- **MD5,**
- **SHA-1, SHA-256,...**
- **....**

all use (variants of) **Merkle-Damgård** transformation.

Hash functions can also be constructed using the number theory.

# MD5 (Message-Digest Algorithm 5)

- **output length**: **128 bits**,
- **designed** by **Rivest** in **1991**,
- in **1996**, **Dobbertin** found collisions in the compresing function of **MD5**,
- in **2004** a group of **Chinese mathematicians** designed a method for finding collisions in **MD5**,
- there exist a tool that finds collisions in **MD5** with a speed **1 collision / minute (on a laptop-computer)**

Is **MD5** completely broken?

The attack would be practical if the colliding documents “made sense”...

In **2005** **A. Lenstra, X. Wang, and B. de Weger** found **X.509** certificates with different public keys and the same **MD5** hash.

# SHA-1 (Secure Hash Algorithm)

- **output length**: **160 bits**,
- designed in **1993** by the **NSA**,
- in **2005** **Xiaoyun Wang, Andrew Yao and Frances Yao** presented an attack that runs in time $2^{63}$.
- Still rather secure, but new hash algorithms are needed!

A US **National Institute of Standards and Technology** is currently running a competition for a new hash algorithm.

# Applications: Online Bid Example

- Suppose Alice, Bob, Charlie are bidders
- Alice plans to bid A, Bob B and Charlie C
  - They do not trust that bids will be secret
  - Nobody willing to submit their bid
- Solution?
  - Alice, Bob, Charlie submit **hashes** h(A),h(B),h(C)
  - All hashes received and posted online
  - Then bids A, B and C revealed
- Hashes do not reveal bids (which property?)
- Cannot change bid after hash sent (which property?)

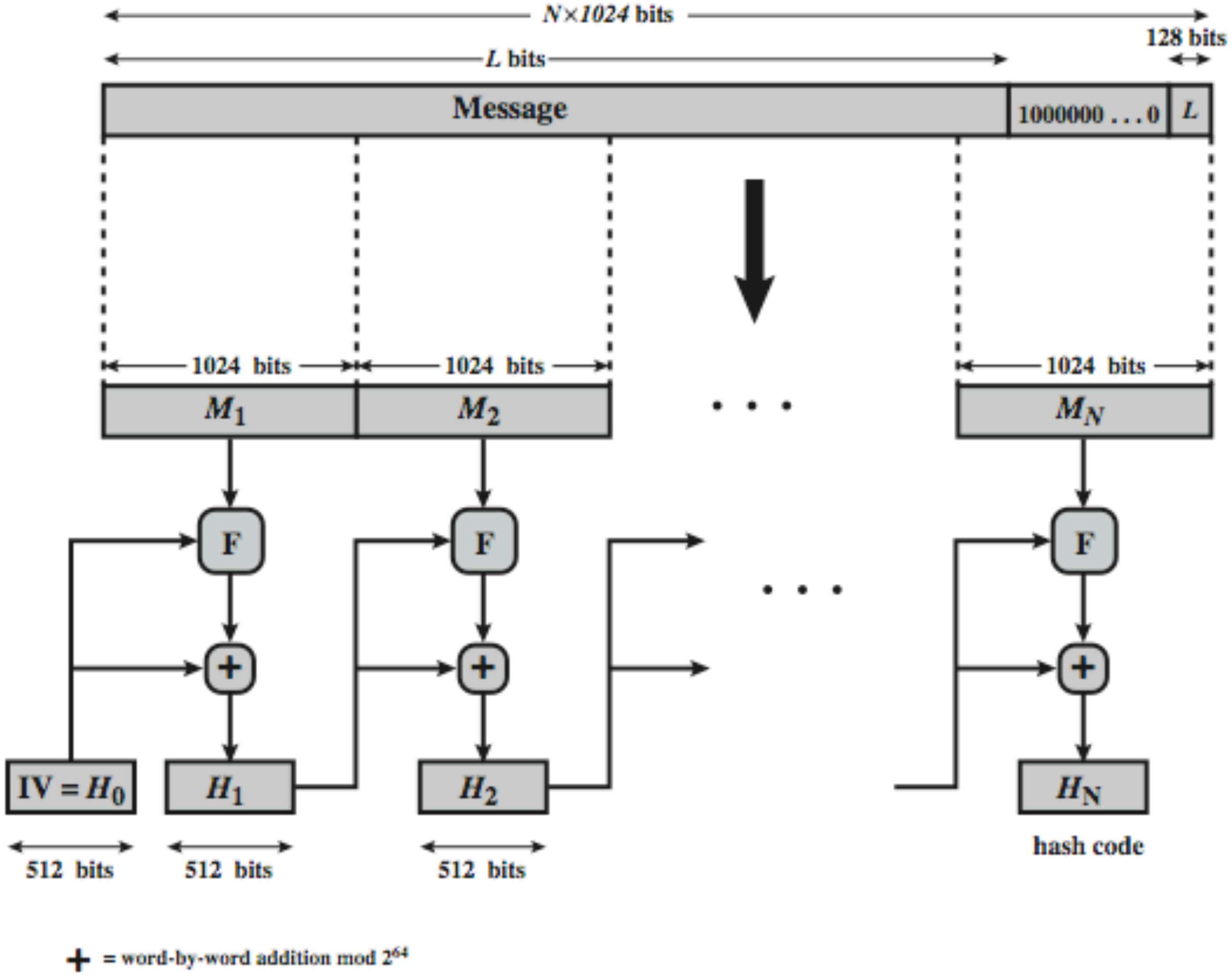
# Online Bid

- This protocol is not secure!
- A forward search attack is possible
  - Bob computes h(A) for likely bids A
- How to prevent this?
- Alice computes h(A,R), R is random
  - Then Alice must reveal A and R
  - Bob cannot try all A and R

# Applications: Securing storage

- Bob has files f1,f2,…,fn
- Bob sends to Amazon S3 the hashes
  - h(r||f1),h(r||f2),…,h(r||fn)
  - The files f1,f2,…,fn
- Bob stores randomness r (and keeps it secret)
- Every time Bob **reads** a file f1, he also reads h(r||fi) and verifies
- Any problems with **writes**?

# SHA-2 overview



# What the industry says about the "hash and authenticate" method?

the block cipher is still there...

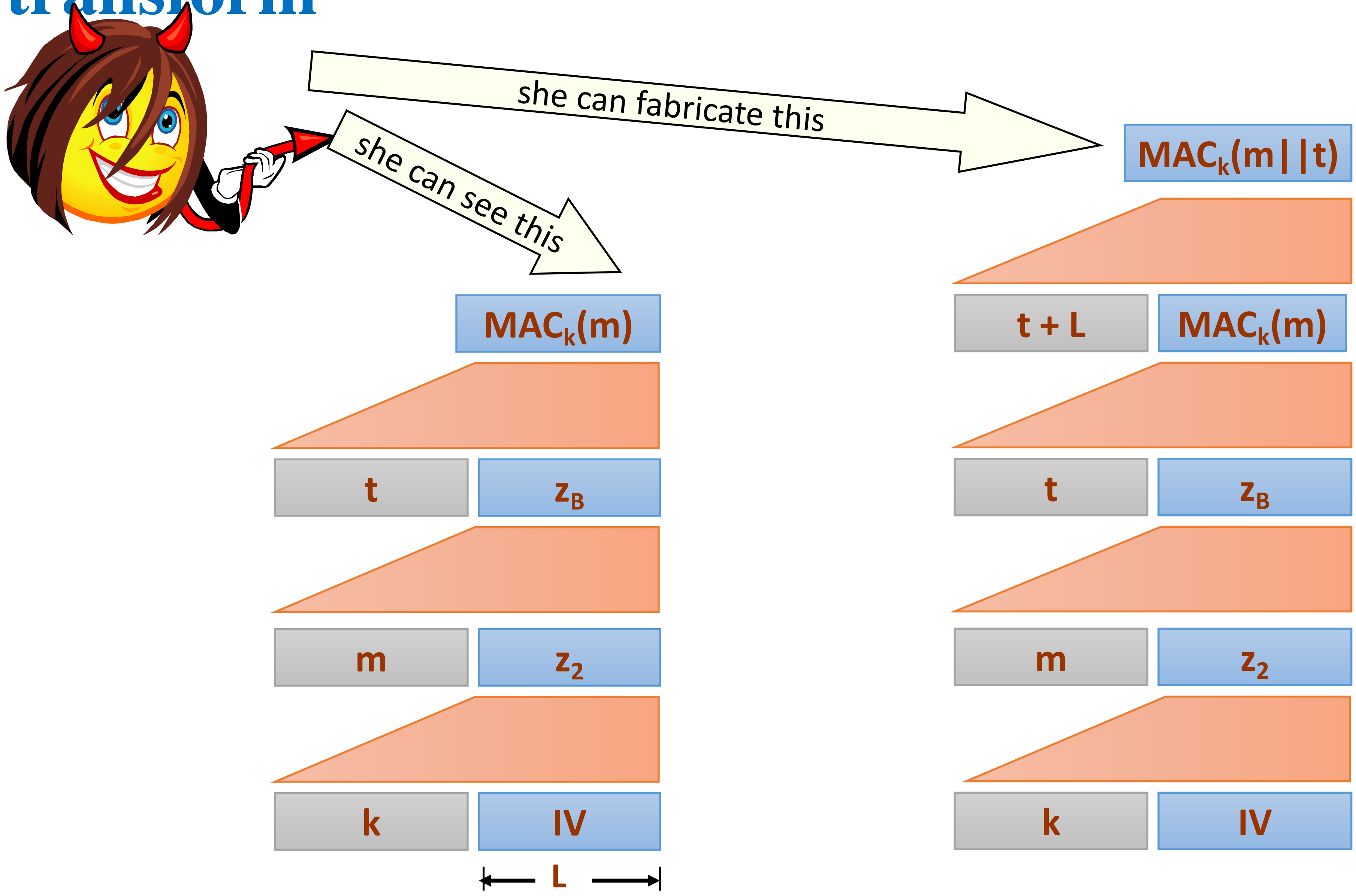
Why don't we just hash a message together with a key:

$MAC_k(m) = H(k \mid\mid m)$

?

**It's not secure!**

# Suppose H was constructed using the MD-transform



# A better idea

**M. Bellare, R. Canetti, and H. Krawczyk (1996):**

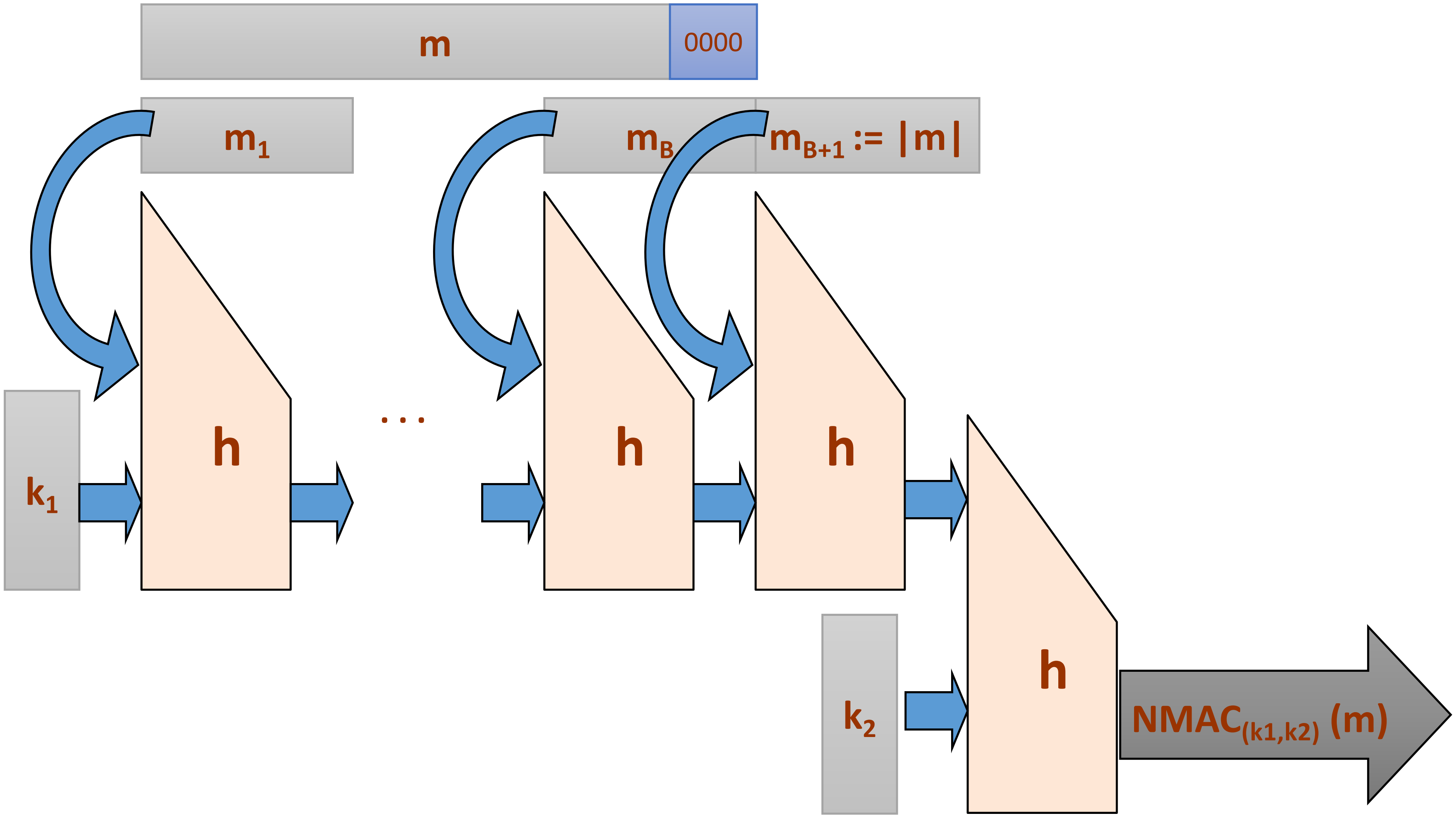
- **NMAC** (Nested MAC)
- **HMAC** (Hash based MAC)

have some "provable properties"

They both use the **Merkle-Damgård** transform.

Again, let **h : $\{0,1\}^{2L} \rightarrow \{0,1\}^{L}$** be a compression function.

# NMAC

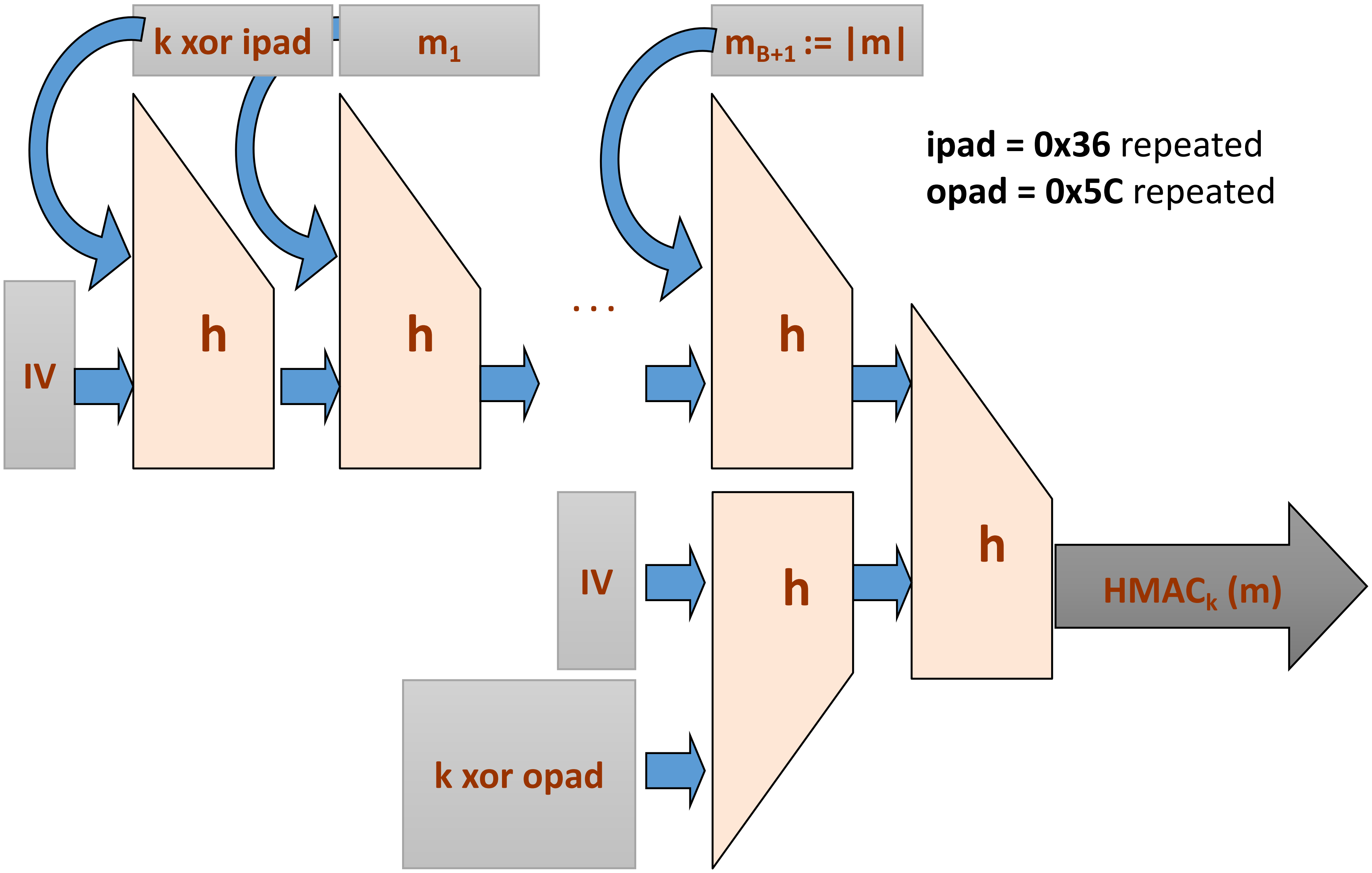


**Looks better, but**

1. our libraries do not permit to change the **IV**
2. the key is too long: **($k_1$,$k_2$)**

**HMAC** is the solution!

# HMAC


k xor ipad
m1
mB+1 := |m|
ipad = 0x36 repeated
opad = 0x5C repeated
IV
h
h
. . .
h
IV
h
h
HMACk (m)
k xor opad

# HMAC – the properties

Looks **complicated**, but it is very easy to implement (given an implementation of **H**):

$$\mathbf{HMAC_k(m) = H((k\ xor\ opad)\ \|\ H(k\ xor\ ipad\ \|\ m))}$$

It has some "provable properties" (slightly weaker than **NMAC**).

Widely used in practice.