ENEE 457: Computer Systems Security
09/10/18

Lecture 4
Symmetric Crypto II

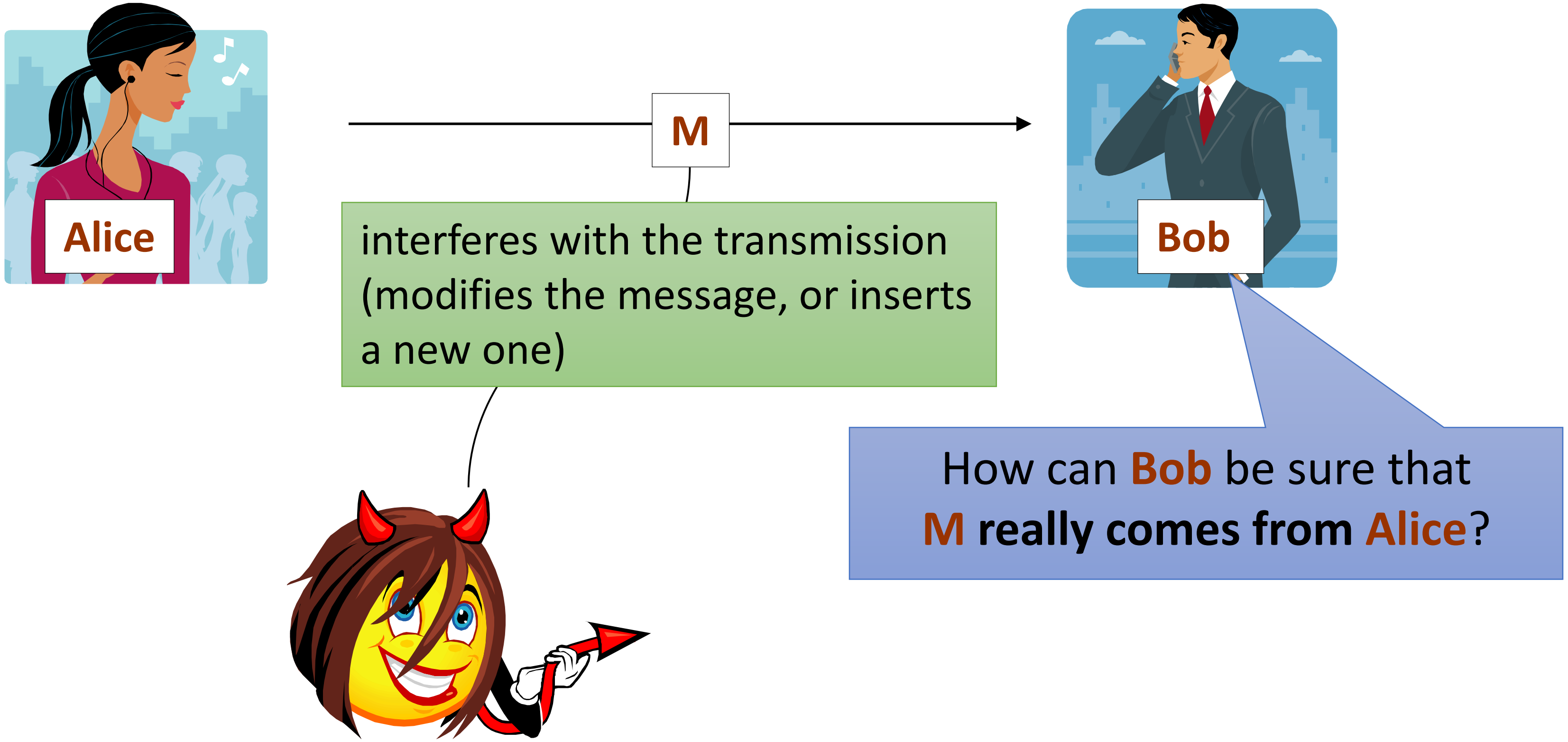
Charalampos (Babis) Papamanthou

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

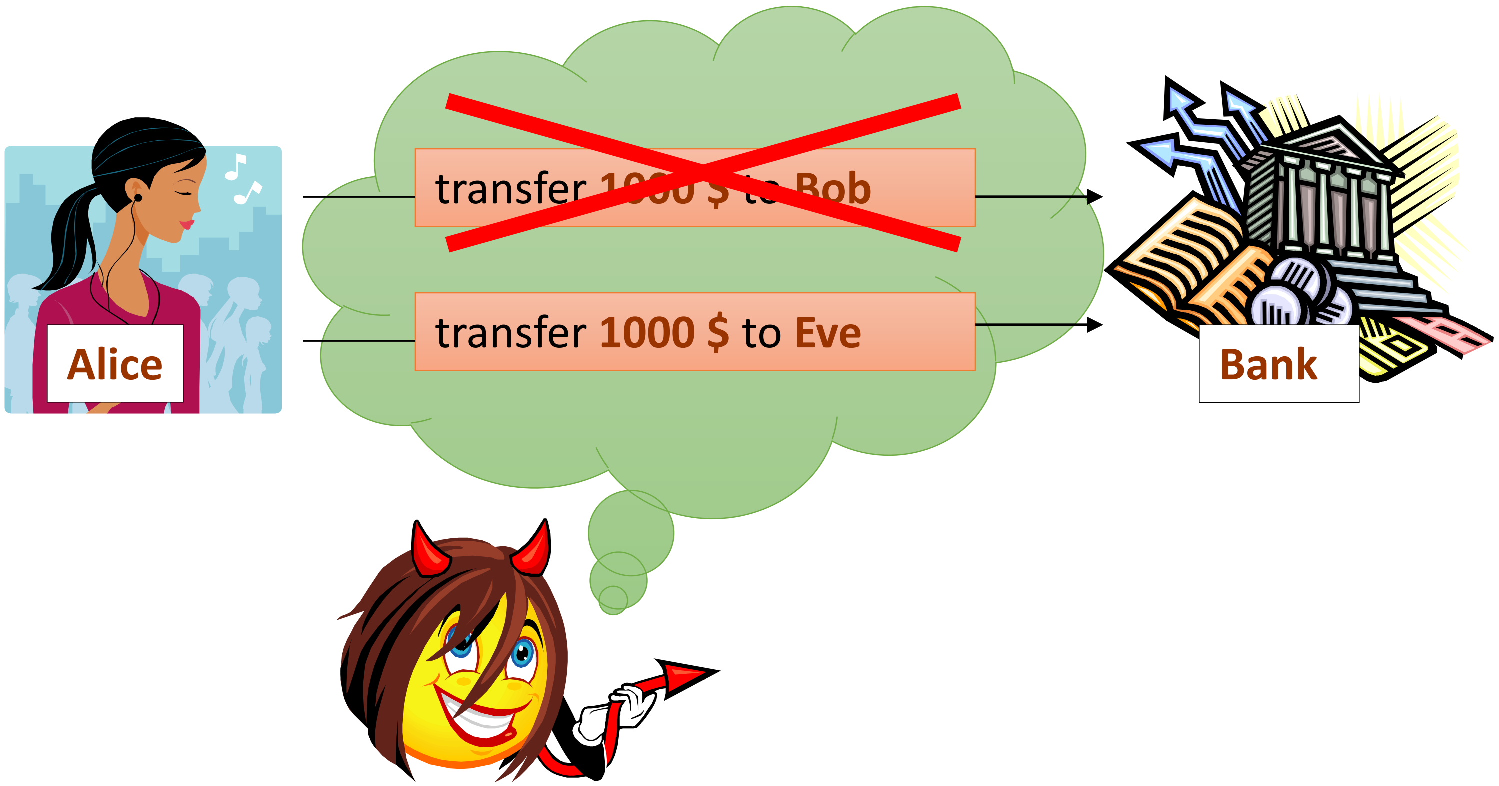
- Slides adjusted from:
  - http://dziembowski.net/Teaching/BISS09/

Message Authentication

Integrity:



Sometimes: more important than secrecy!



Of course: usually we want both **secrecy** and **integrity**.

Does encryption guarantee message integrity?

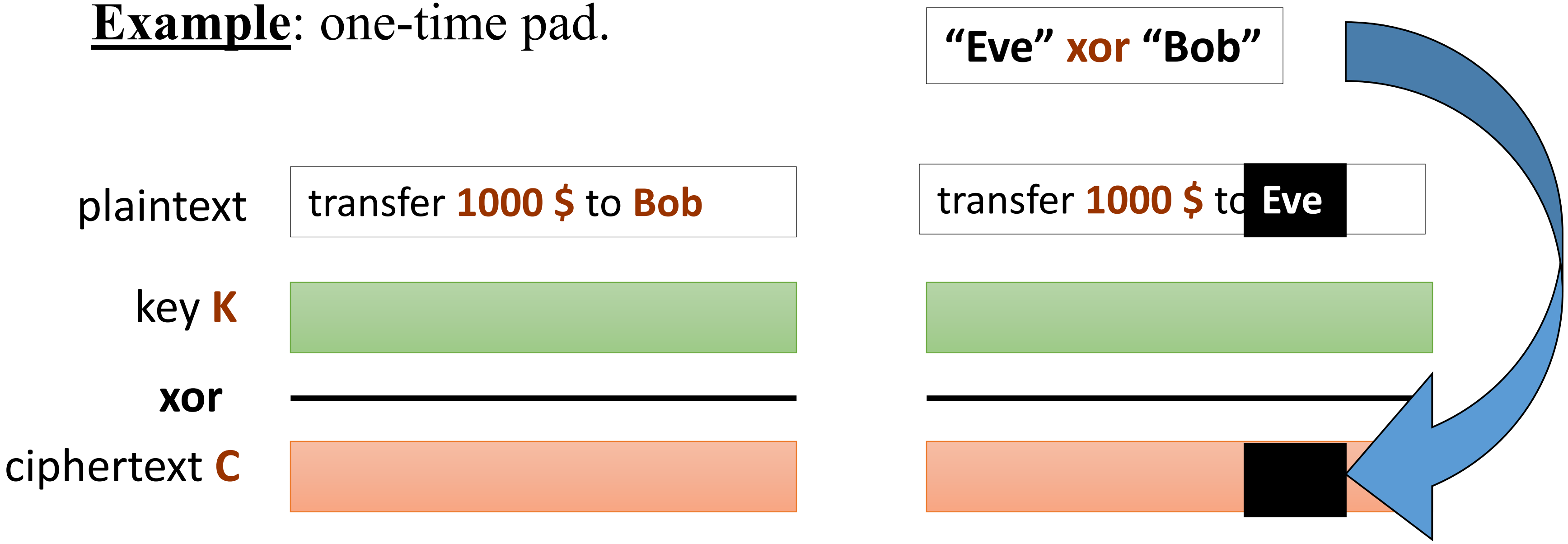
Idea:

1. Alice encrypts **m** and sends **c=Enc(k,m)** to Bob.
2. Bob computes **Dec(k,m),** and if it "*makes sense*" **accepts it**.

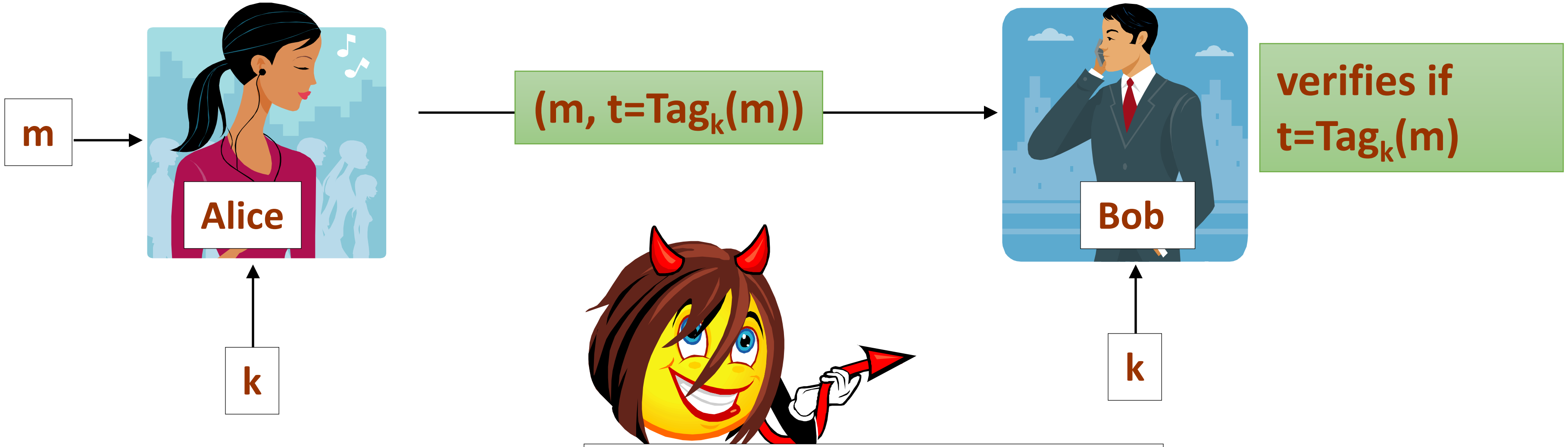
Intuiton: only Alice knows **k**, so nobody else can produce a valid ciphertext.

It does not work!

Example: one-time pad.



Message authentication



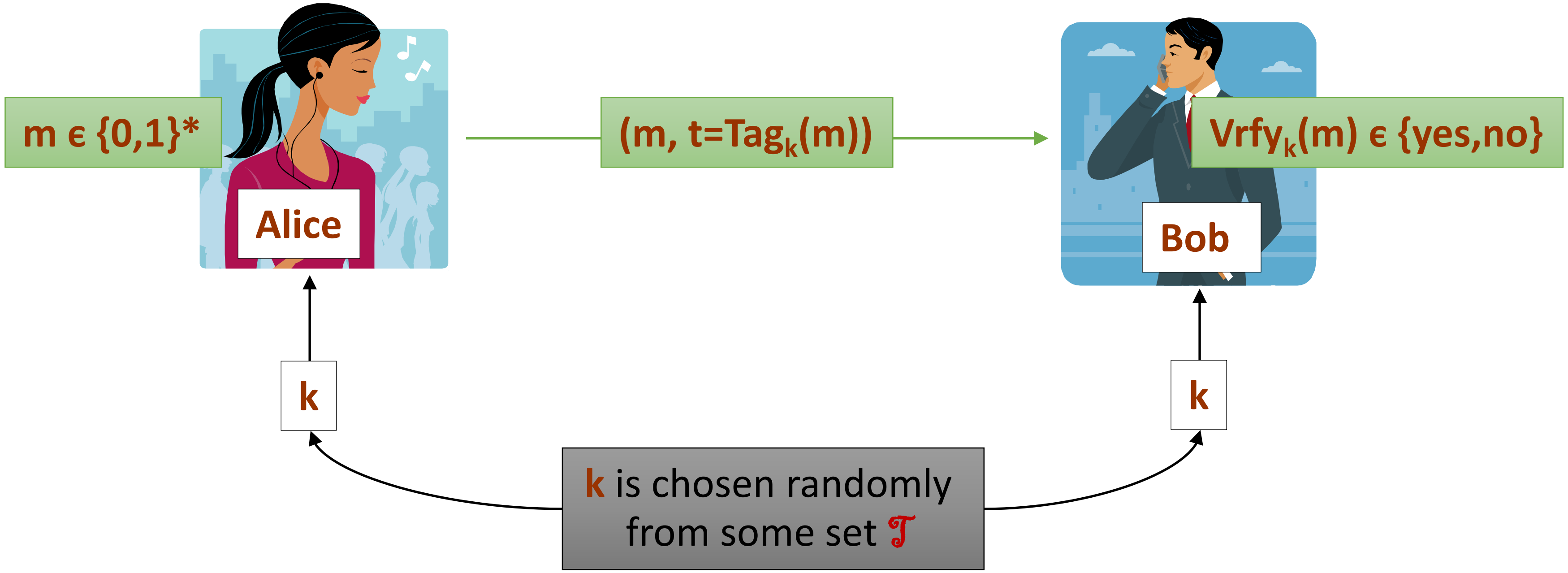
Eve can see **$(m, t=Tag_k(m))$**

She should not be able to compute a valid tag **t’** on any other message **m’**.

Message authentication – multiple messages

Eve should not be able to compute a valid tag **t'** on any other message **m'**.

Message Authentication Codes – the idea



A mathematical view

$\mathcal{K}$ – **key** space

$\mathcal{M}$ – **plaintext** space

$\mathcal{T}$ - set of **tags**

A **MAC scheme** is a pair **(Tag, Vrfy)**, where

- Tag : $\mathcal{K} \times \mathcal{M} \rightarrow \mathcal{T}$ is an tagging algorithm,
- **Ver:** $\mathcal{K} \times \mathcal{M} \times \mathcal{T} \rightarrow$ **{yes, no}** is an **decryption** algorithm.

We will sometimes write $\mathbf{Tag_k(m)}$ and $\mathbf{Vrfy_k(m,t)}$ instead of **Tag(k,m)** and **Vrfy(k,m,t)**.

Correctness

it should always holds that:

$\mathbf{Vrfy_k(m,Tag_k(m)) = yes}$.

How to define security?

We need to specify:

1. how the messages $m_1, \dots, m_w$ are chosen,

2. what is the goal of the adversary.

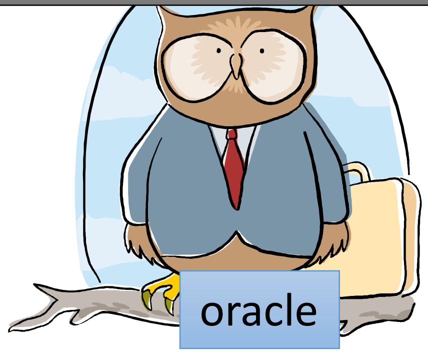
Good tradition: be as pessimistic as possible!

Therefore we assume that

1. The adversary is allowed to chose $m_1, \dots, m_w$.
2. The goal of the adversary is to produce a valid tag on some m' such that $m' \neq m_1, \dots, m_w$.

adversary

selects random a $k \in \{0,1\}^n$



oracle

m_1 →

← $(m_1, t=Tag_k(m_1))$

⋮

m_w →

← $(m_w, t=Tag_k(m_w))$

We say that **the MAC scheme is secure if** at the end the adversary **cannot output** **(m’,t’)** such that

Vrfy(m’,t’) = yes

and

$m' \neq m_1,\ldots,m_w$

Aren’t we too paranoid?

Maybe it would be enough to require that:

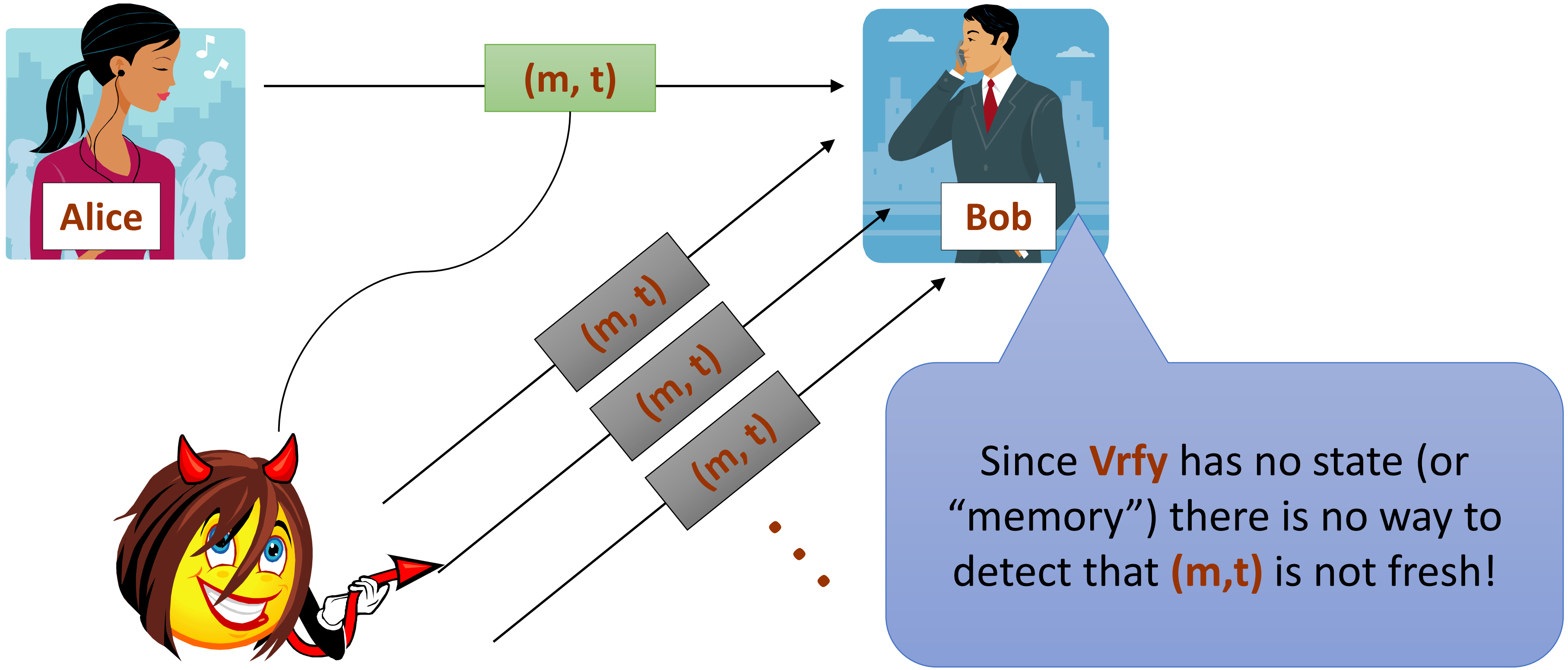
the adversary succeds only if he forges a message that “*makes sense*”.

(e.g.: forging a message that consists of **random noise** should not count)

Bad idea:

- hard to define,
- is application-dependent.

Warning: MACs do not offer protection against the “replay attacks”.



This problem has to be solved by the higher-level application (methods: **time-stamping**, **sequence numbers**...).

Authentication and Encryption

Usually we want to authenticate and encrypt at the same time.

What is the right way to do it? There are several options:

- **Encrypt-and-authenticate:**

  $\mathbf{c \leftarrow Enc_{k1}(m)}$ and $\mathbf{t \leftarrow Mac_{k2}(m)}$

  wrong

- **Authenticate-then-encrypt**:

  $\mathbf{t \leftarrow Mac_{k2}(m)}$ and $\mathbf{c \leftarrow Enc_{k1}(m||t)}$

  better

- **Encrypt-then-authenticate**:

  $\mathbf{c \leftarrow Enc_{k1}(m)}$ and $\mathbf{t \leftarrow Mac_{k2}(c)}$

  the best

By the way: **never** use the same key for **Enc** and **Mac**:

$\mathbf{k_1}$ and $\mathbf{k_2}$ have to be "independent"!

Constructing a MAC

1. **MACs** can be constructed from the block-ciphers.
   We will now discuss to constructions:
   - **simple** (and **not practical**),
   - a little bit **more complicated** (and **practical**) – a **CBC-MAC**

1. **MACs** can also be constructed from the hash functions
   (**NMAC**, **HMAC**).

A simple construction from a block cipher

Let

$$F : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$$

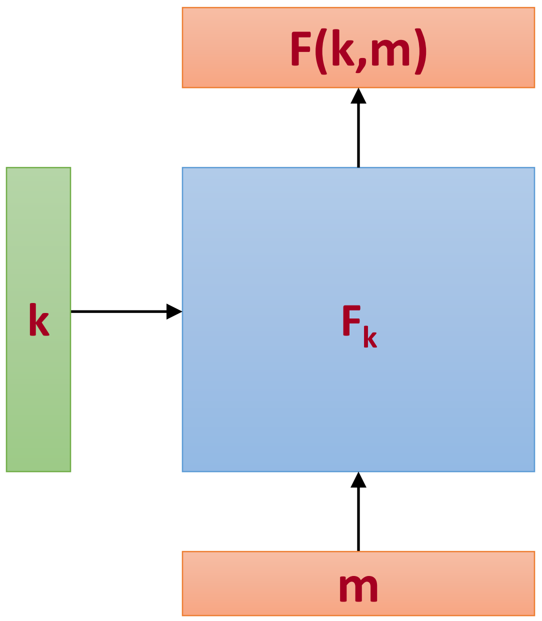
be a **block cipher**.

We can now define a **MAC** scheme that works only for messages $m \in \{0,1\}^n$ as follows:

- $\mathbf{Mac(k,m) = F(k,m)}$

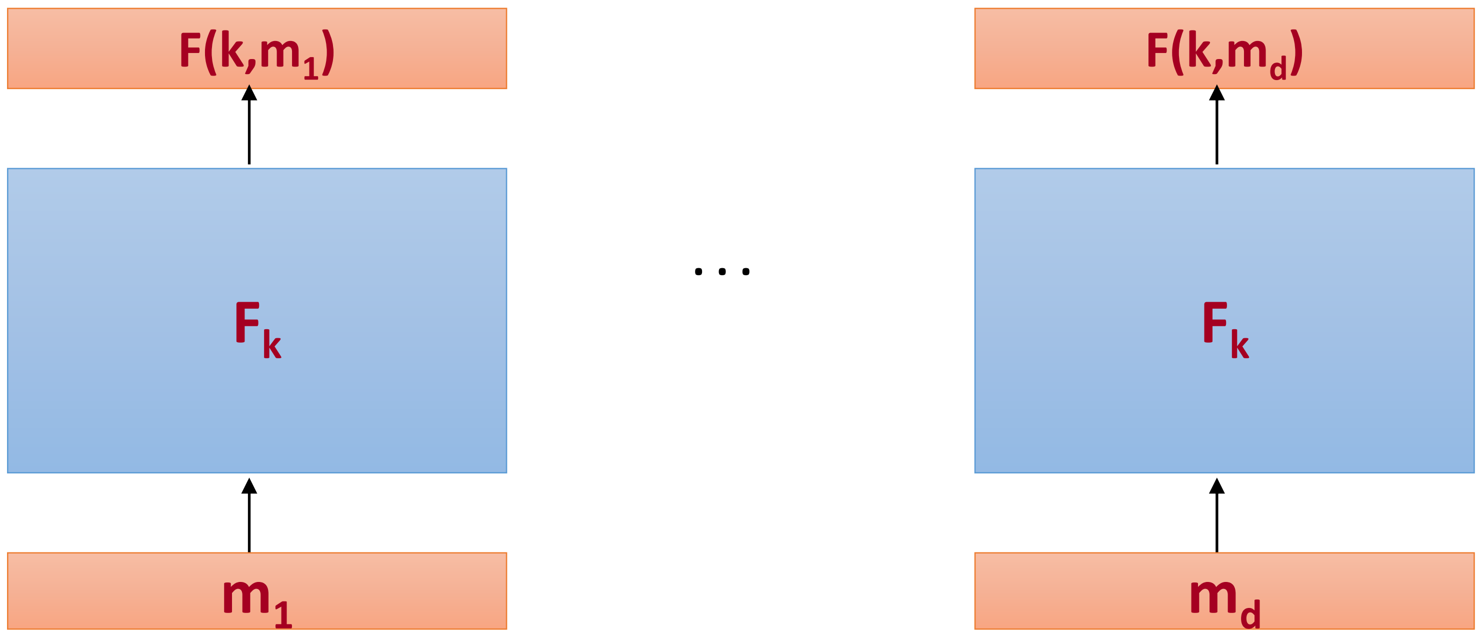
It can be proven that it is a secure **MAC**.

How to generalize it to longer messages?



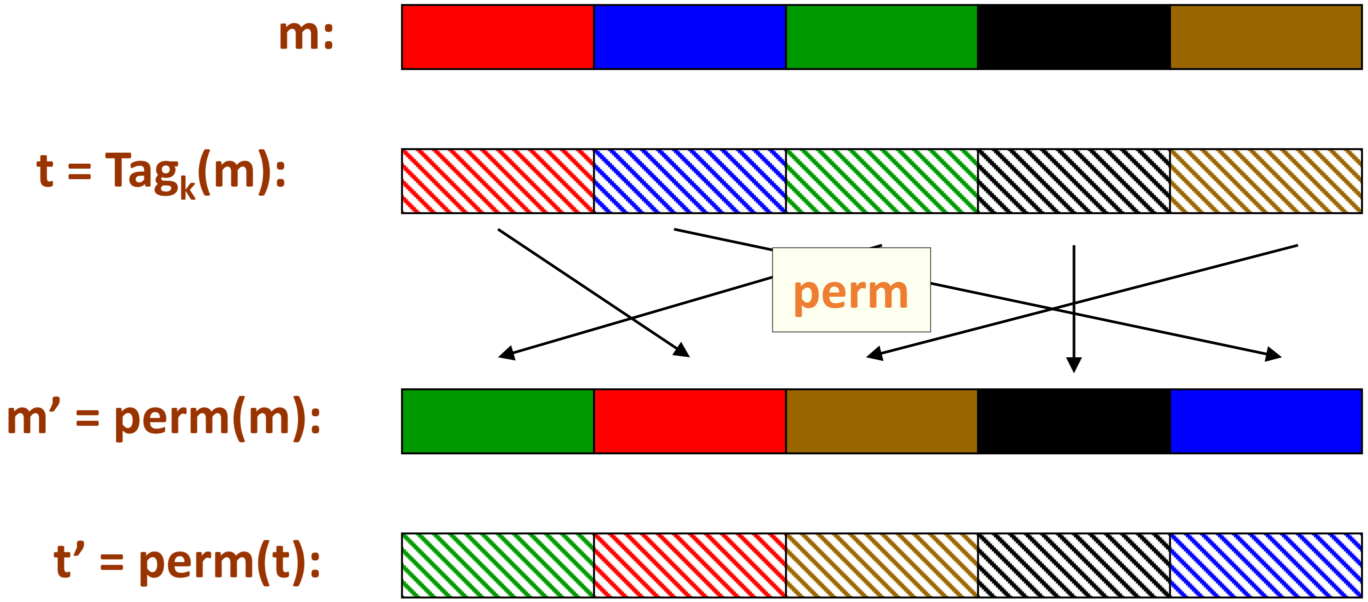
Idea 1

- divide the message in blocks $m_1,\ldots,m_d$
- and authenticate each block separately



This doesn't work!

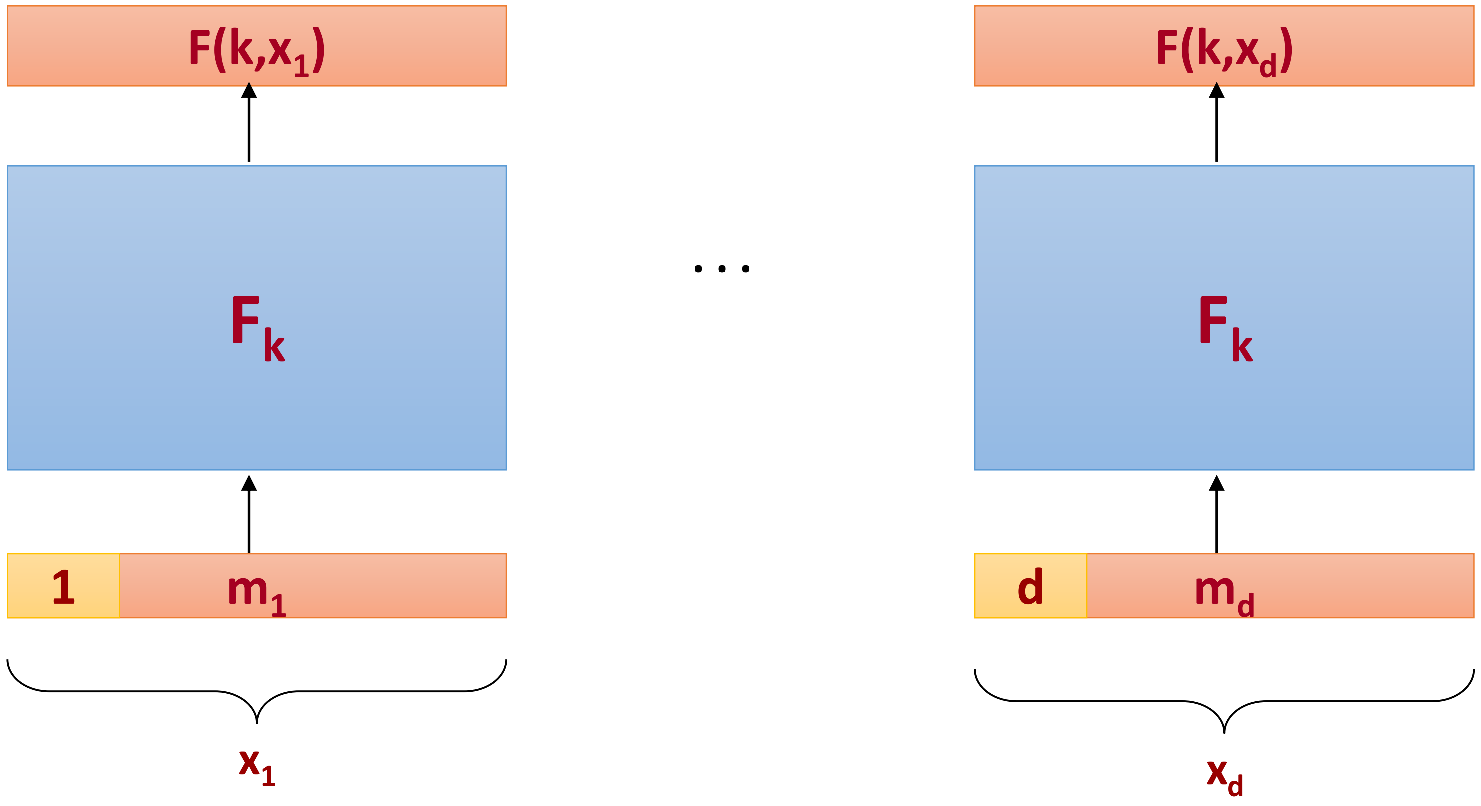
What goes wrong?



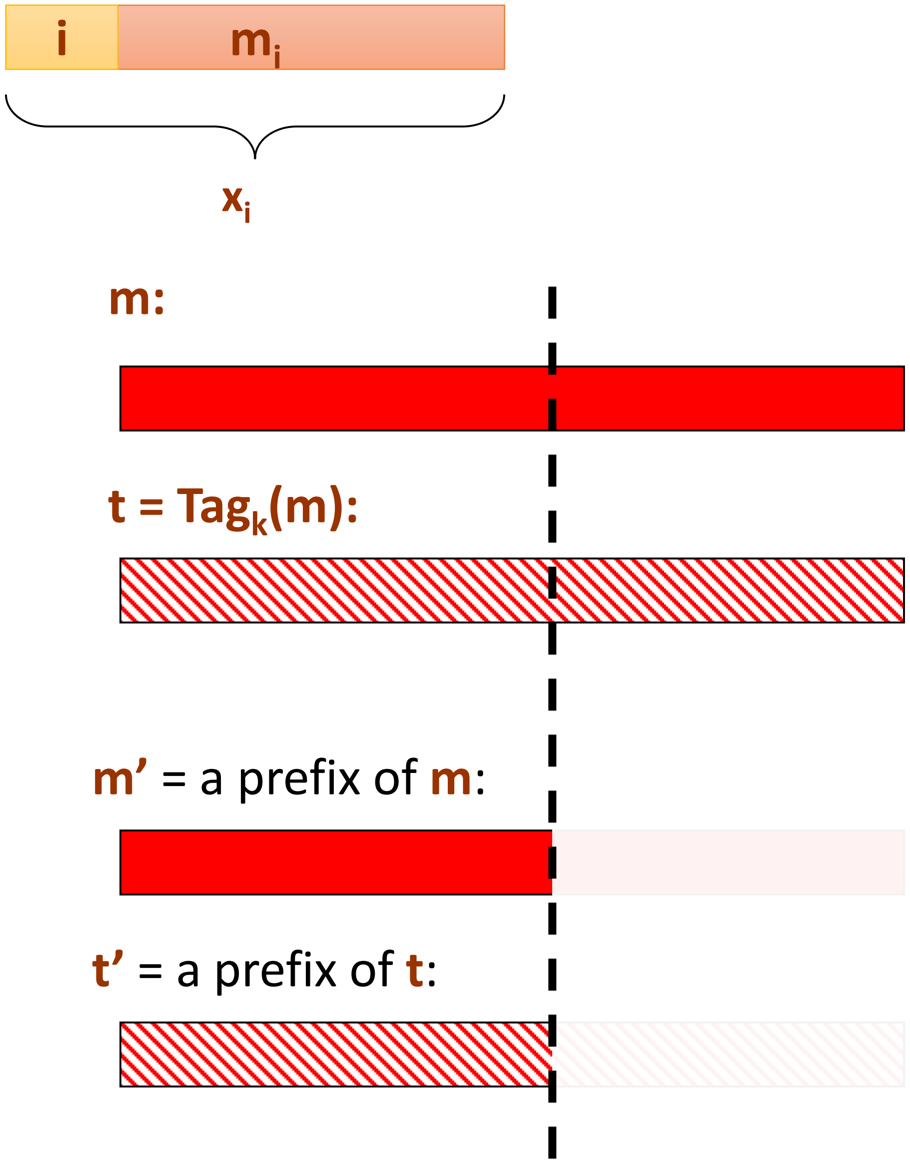
Then **t’** is a valid tag on **m’**.

Idea 2

Add a counter to each block.



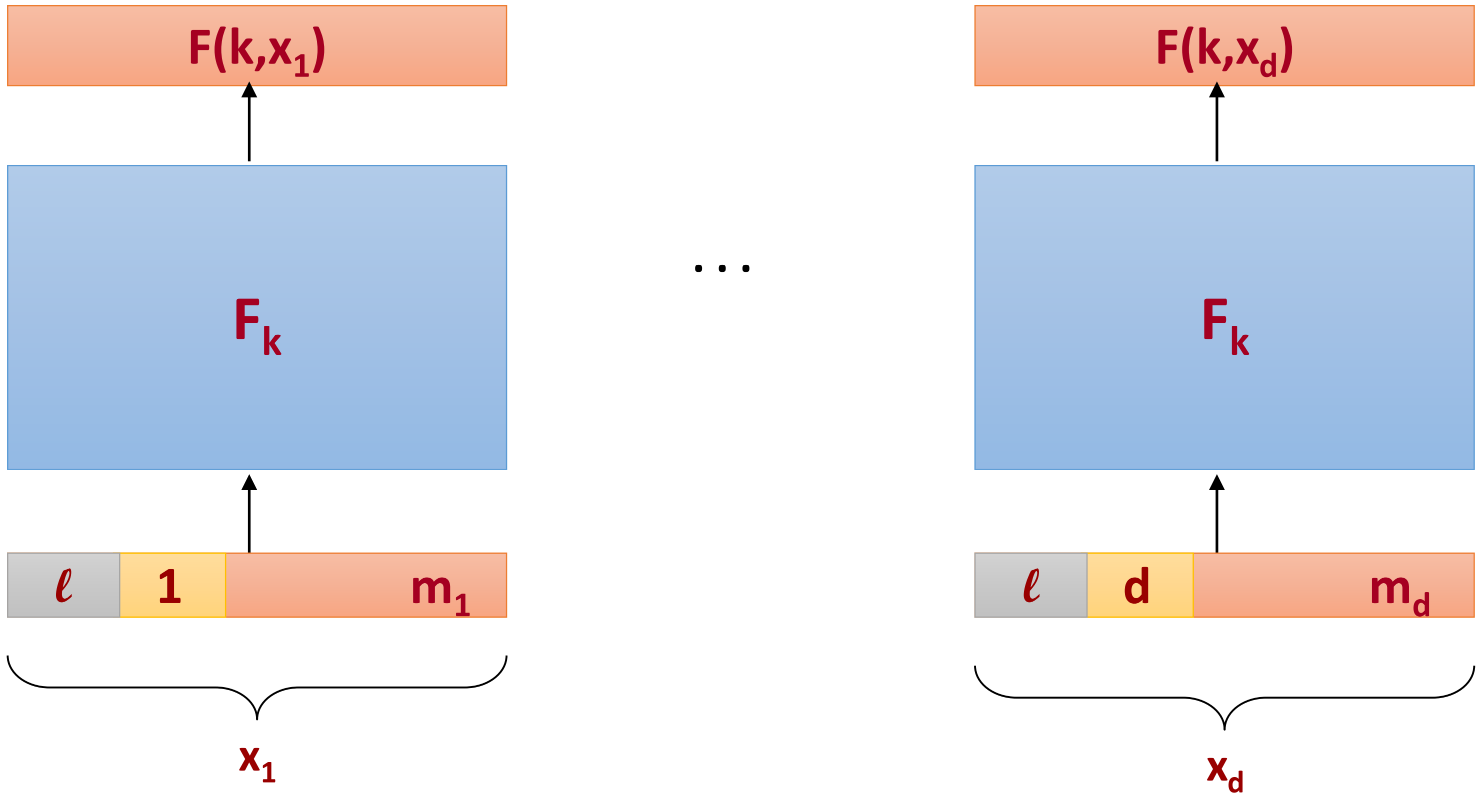
This doesn't work either!



Then **t'** is a valid tag on **m'**.

Idea 3

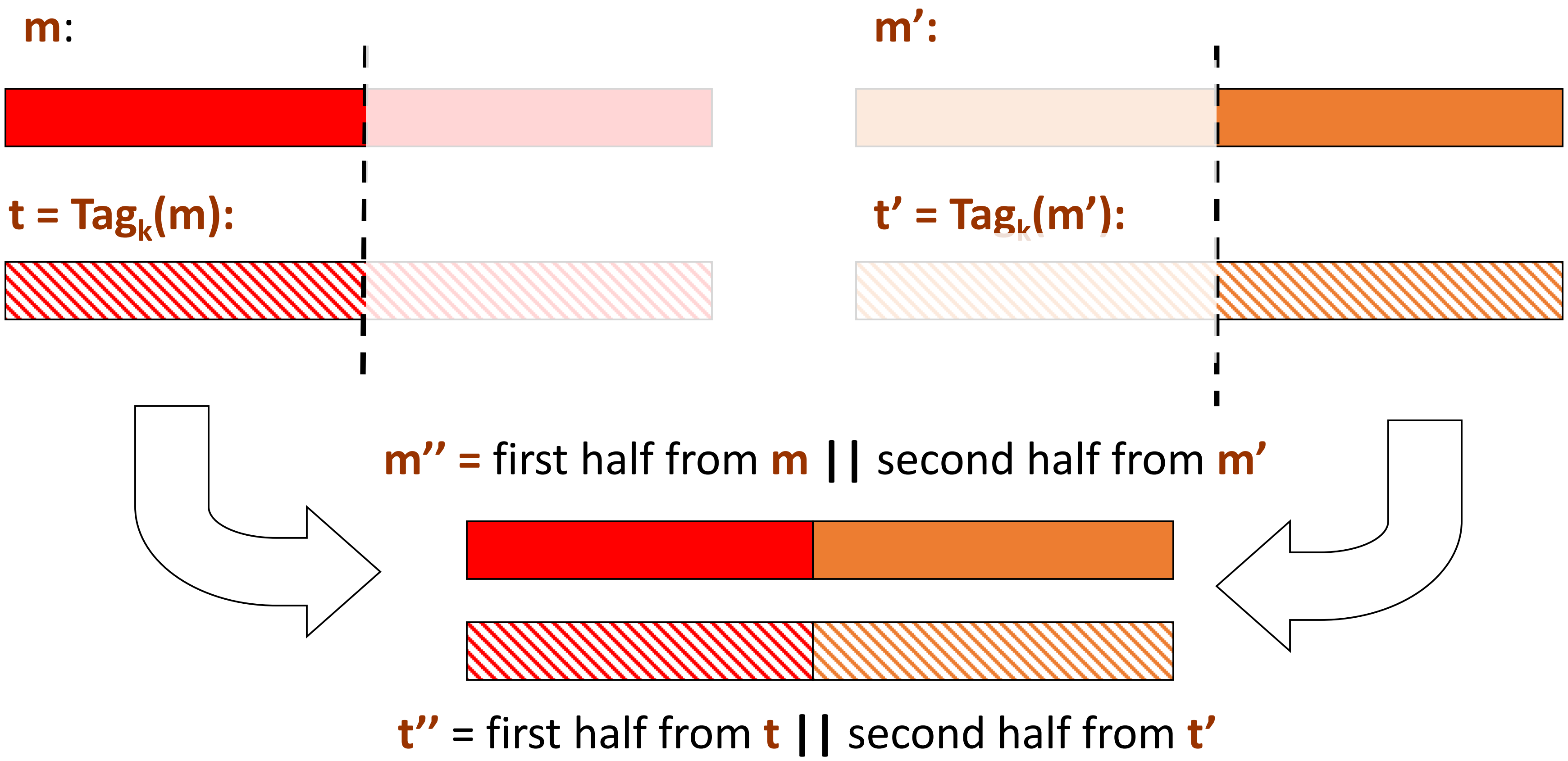
Add ℓ := |m| to each block



This doesn't work either!

$$\underbrace{\ell \;|\; 1 \;|\; m_1}_{x_i}$$

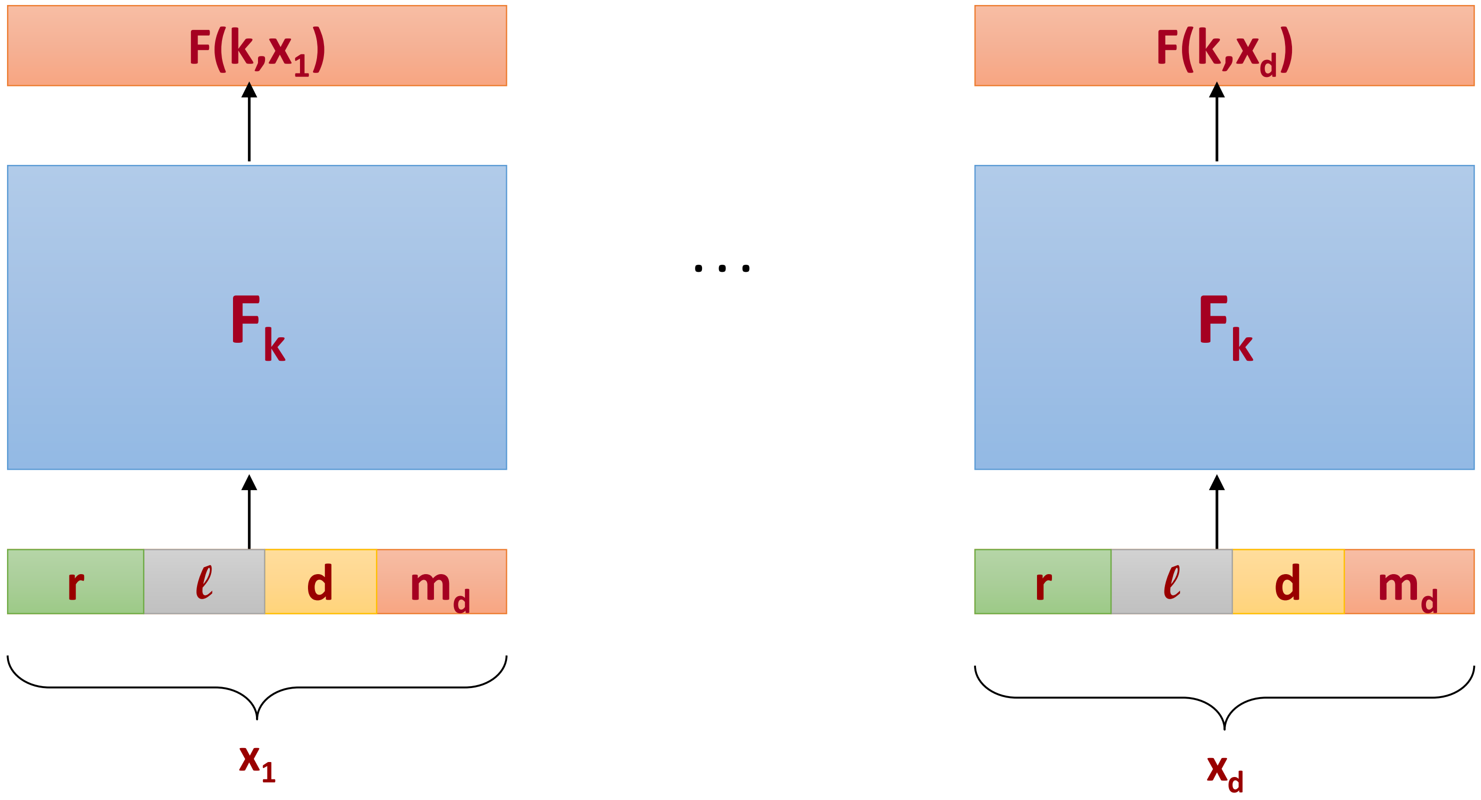
What goes wrong?



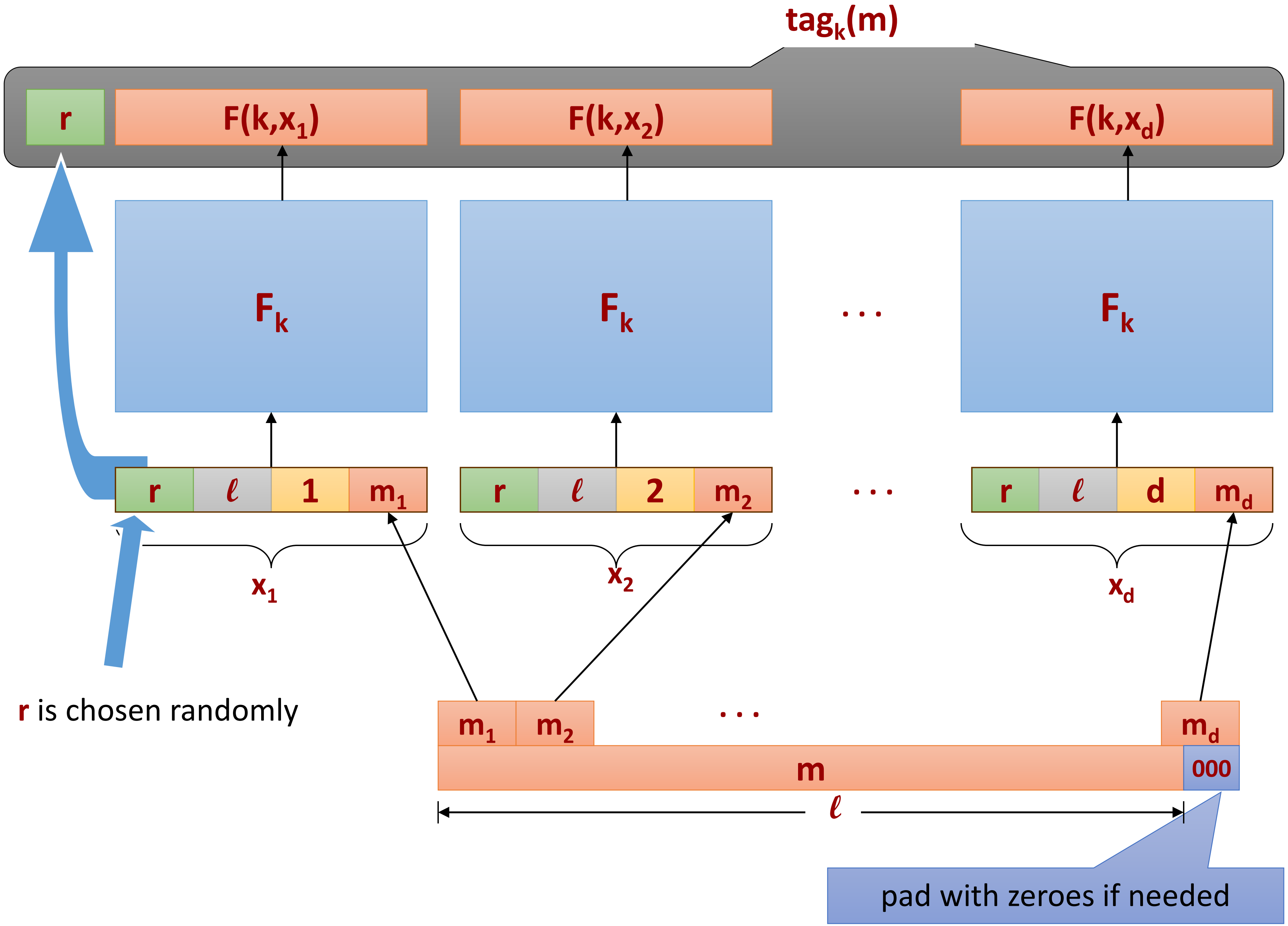
Then **t''** is a valid tag on **m''**.

Idea 4

Add a fresh random value to each block!



This works!



$tag_k(m)$
r
$F(k,x_1)$
$F(k,x_2)$
$F(k,x_d)$
F_k
F_k
. . .
F_k
r
ℓ
1
m_1
r
ℓ
2
m_2
. . .
r
ℓ
d
m_d
x_1
x_2
x_d
r is chosen randomly
m_1
m_2
. . .
m_d
m
000
ℓ
pad with zeroes if needed

This construction can be proven secure

Theorem

Assuming that

$F : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$ is a **pseudorandom permutation**

the construction from the previous slide is a secure **MAC**.

This construction is not practical

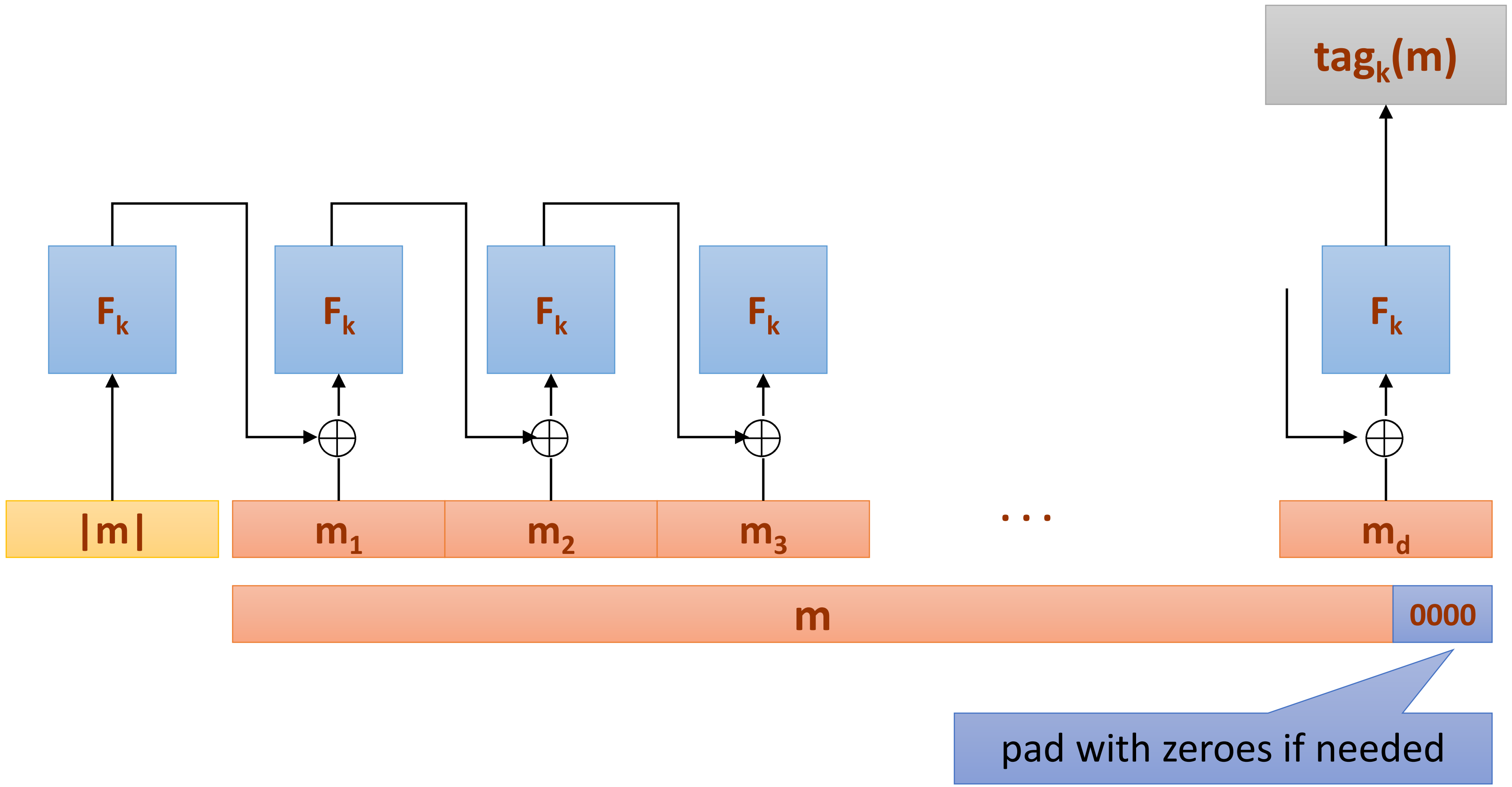
Problem:

The tag is **at least as big as** the message...
But we do not need to decrypt, just to verify

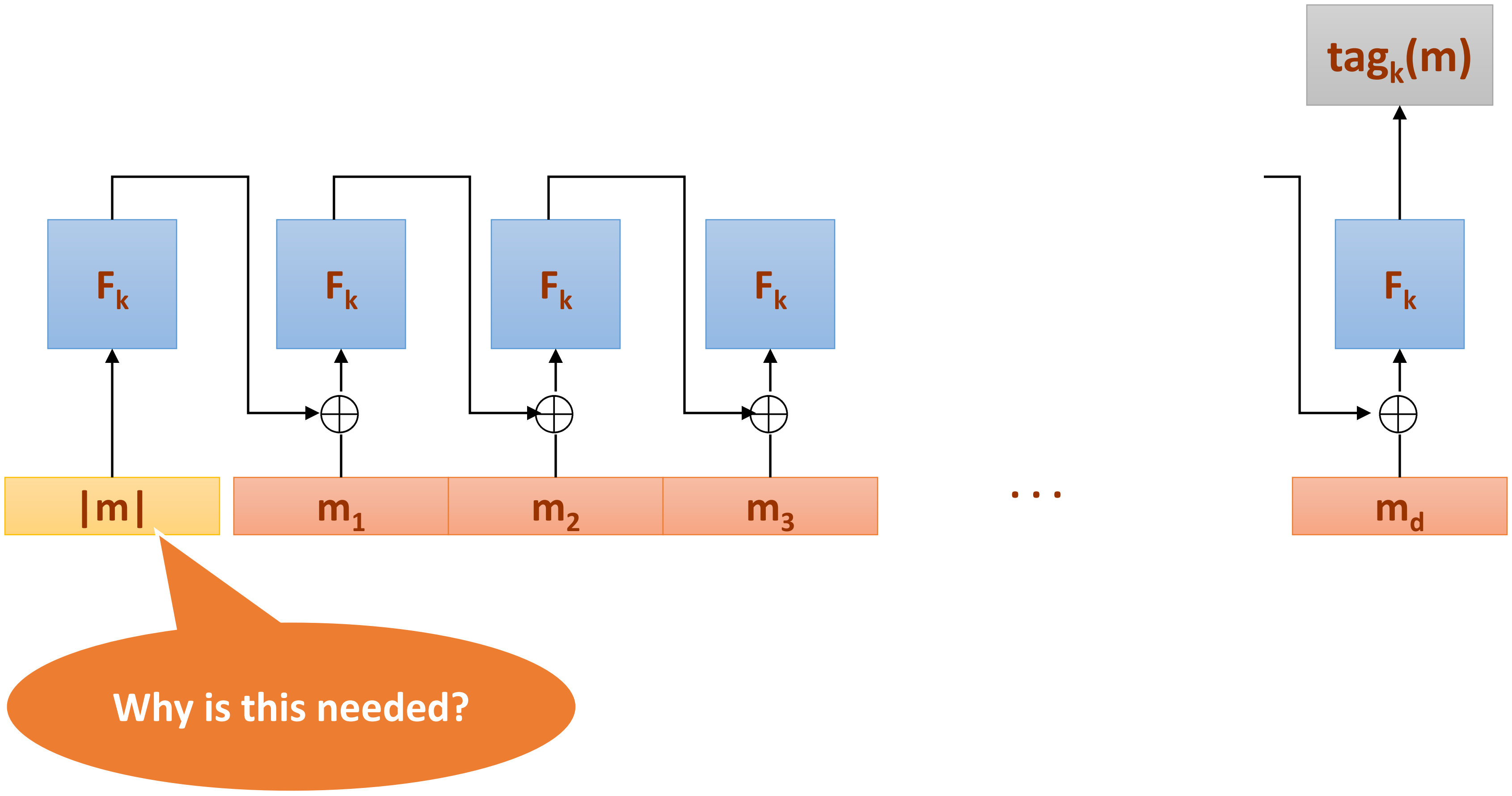
We can do much better!

CBC-MAC

$F : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$ - a block cipher

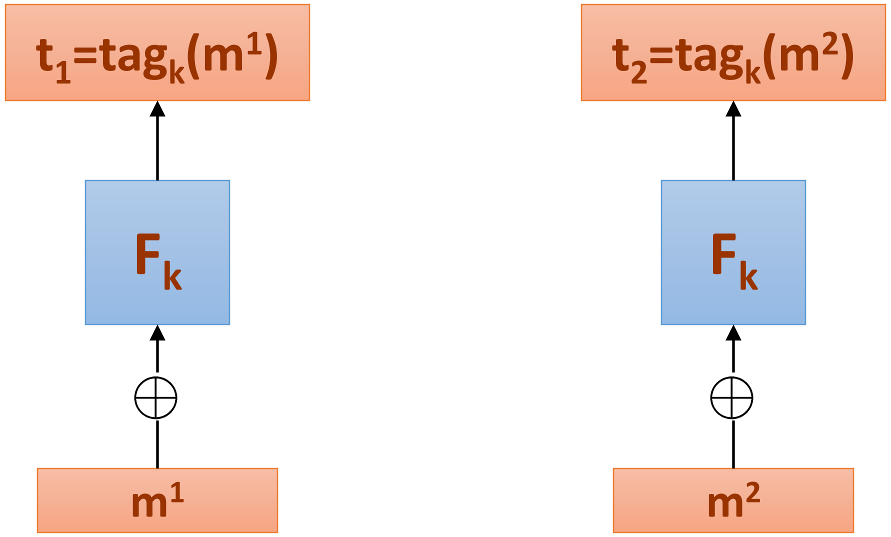


Other variants exist!

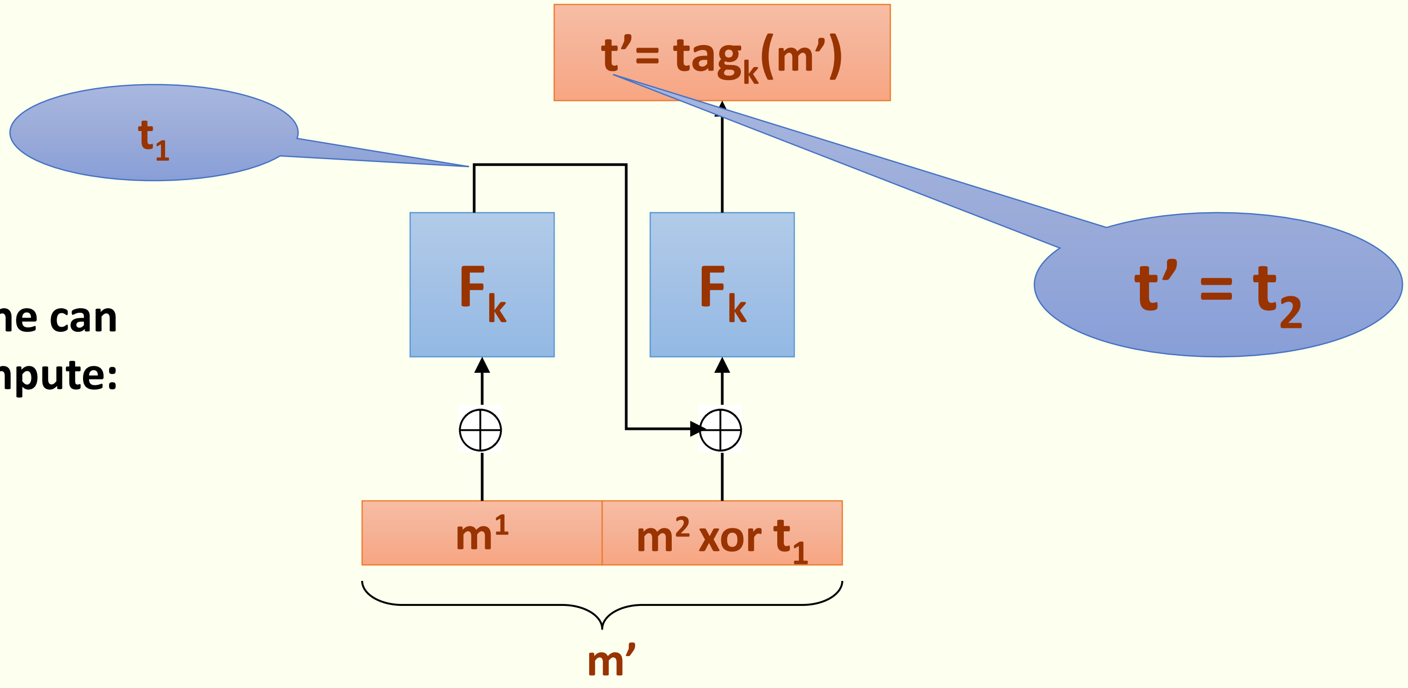


Suppose we do not prepend **|m|**...

the adversary chooses:



now she can compute:



Some practictioners don’t like the CBC-MAC

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

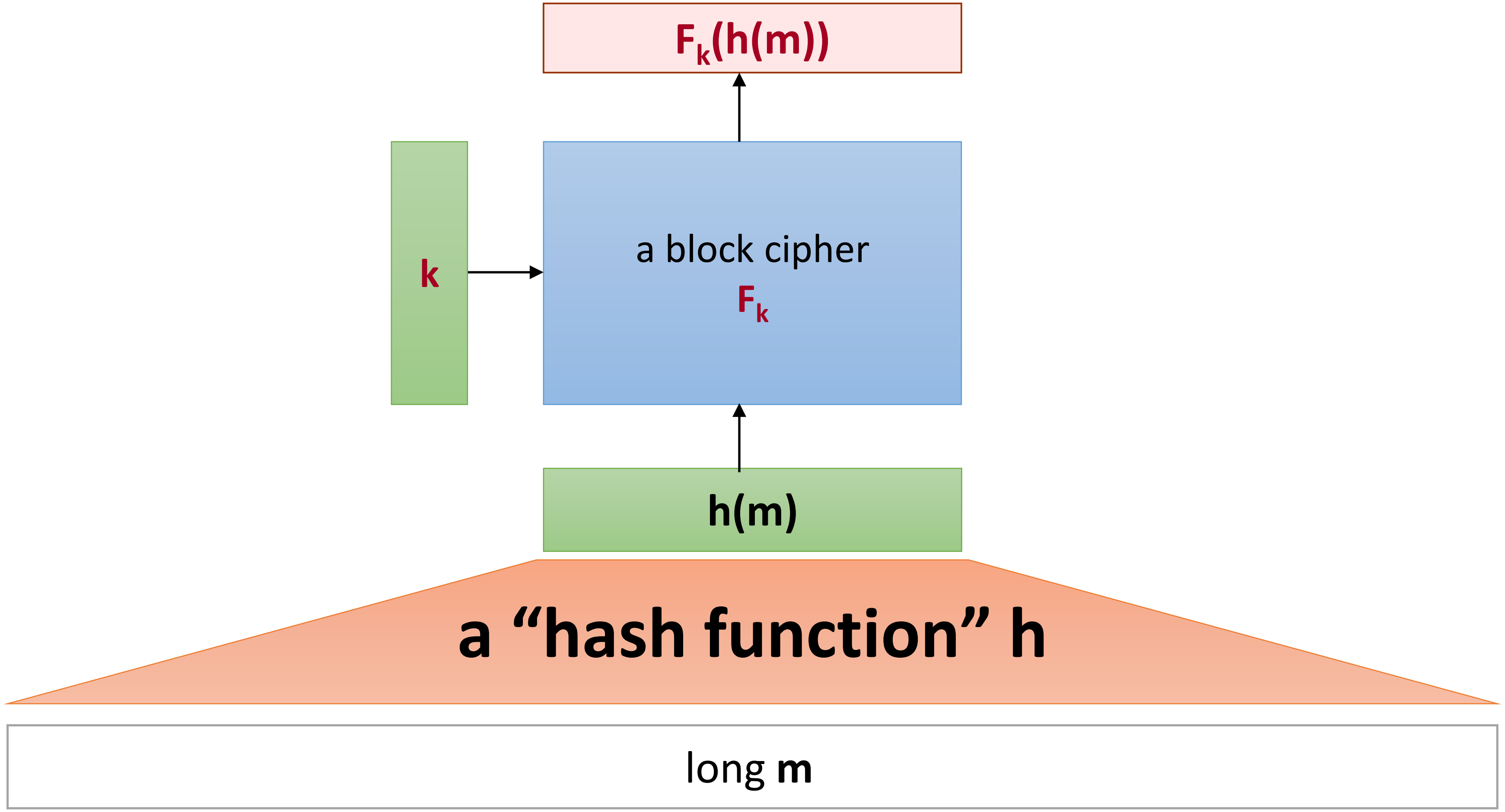
What do you want to use instead?

Hash functions!

Why?

Because they are **more efficient**

Another idea for authenticating long messages



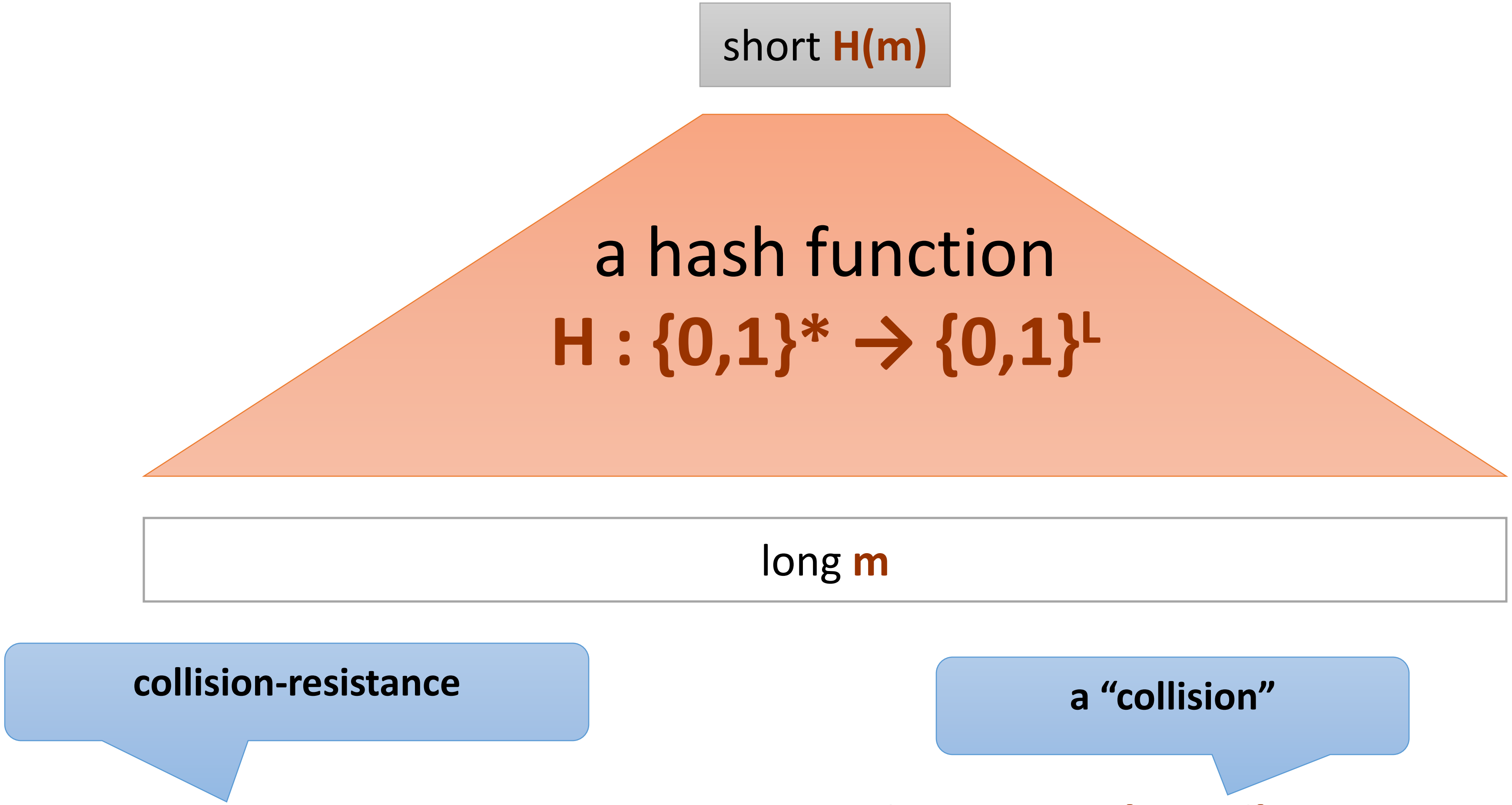
How to formalize it?

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

The basic property that we require is:

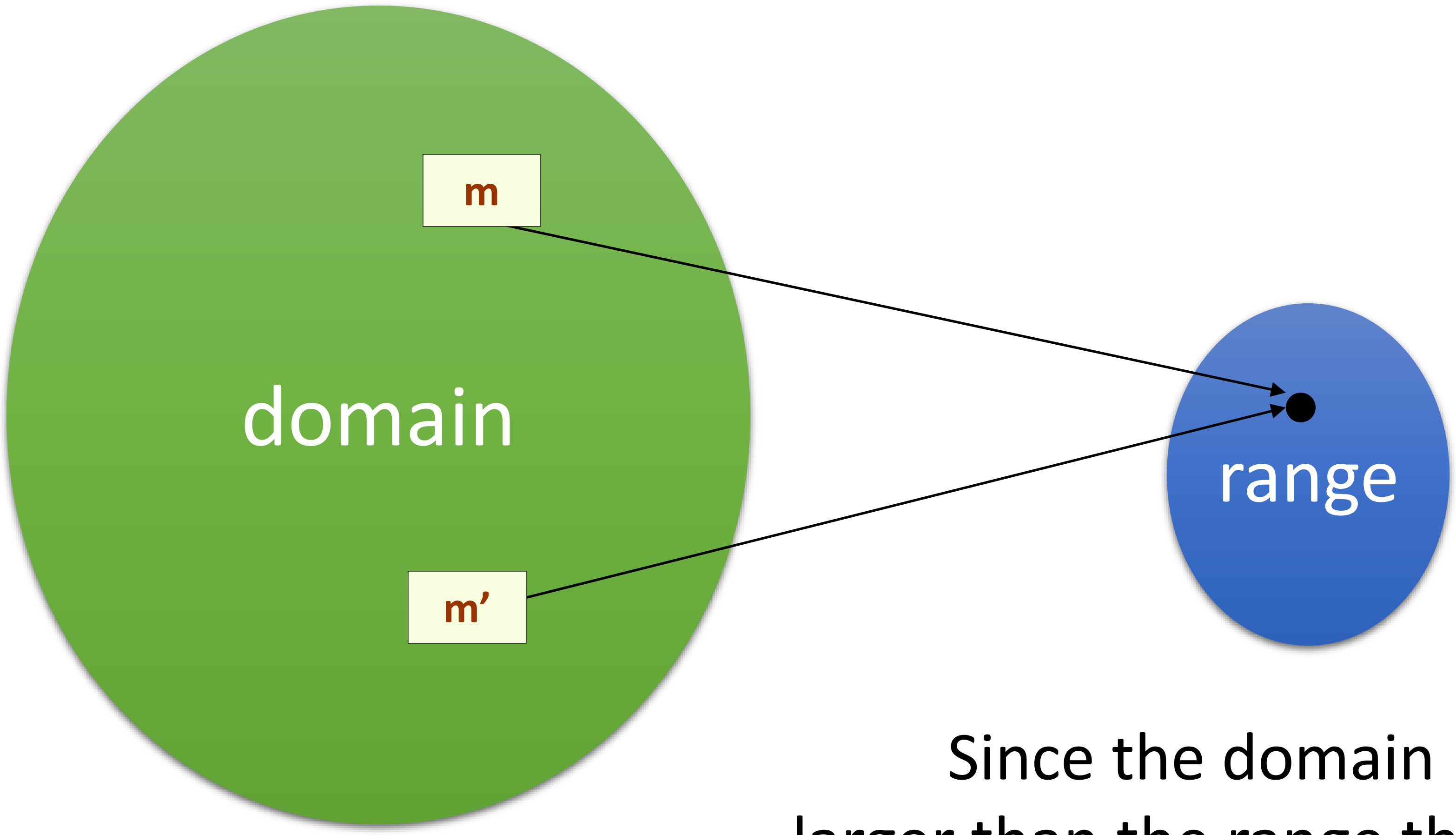
“collision resistance”

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”

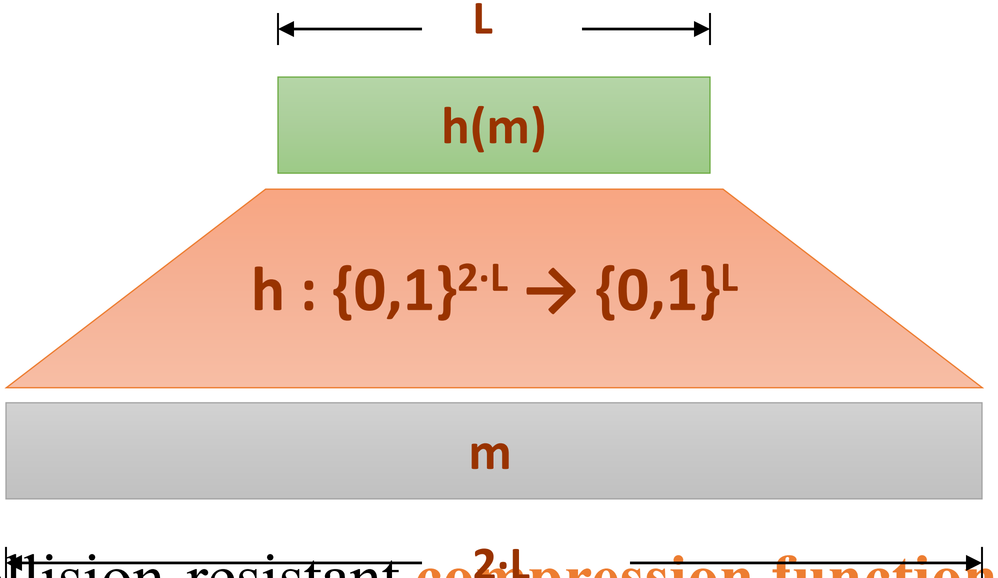
H is a **collision-resistant hash function** if it is “*practically impossible to find collisions in **H***”.

Popular hash funcitons:

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

A common method for constructing hash functions

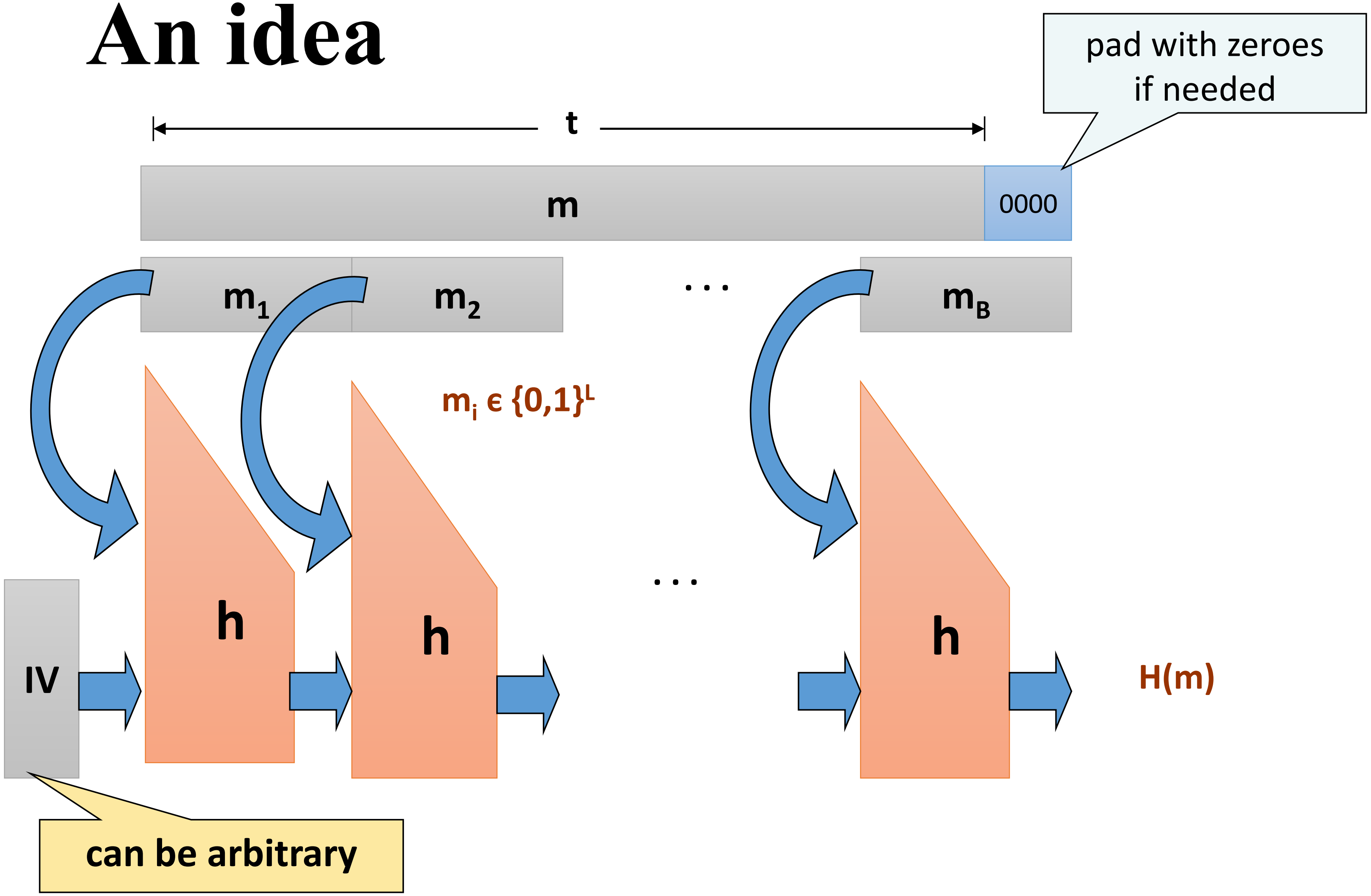
1. Construct a "***fixed-input-length***" collision-resistant hash function



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

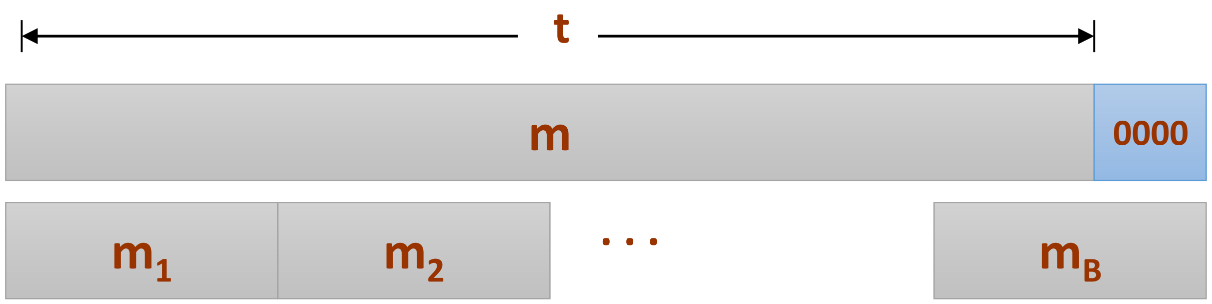
2. Use it to construct a hash function.

An idea



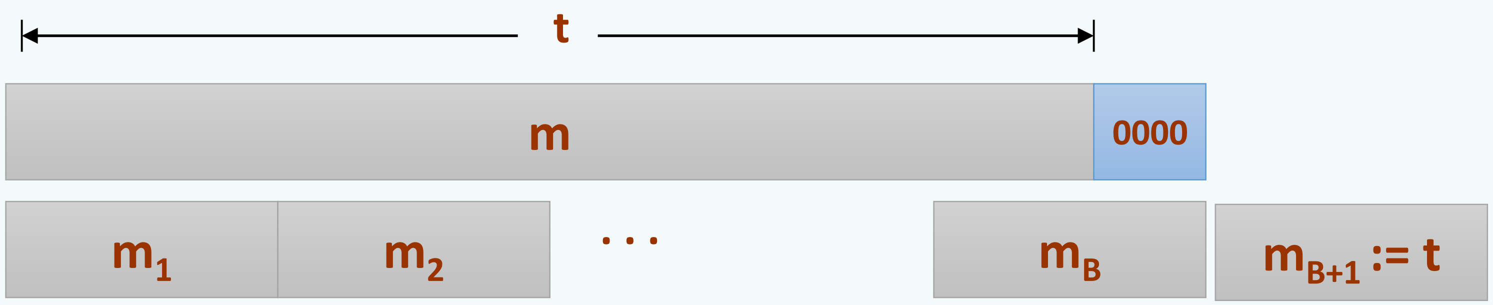
This doesn’t work...

Why is it wrong?



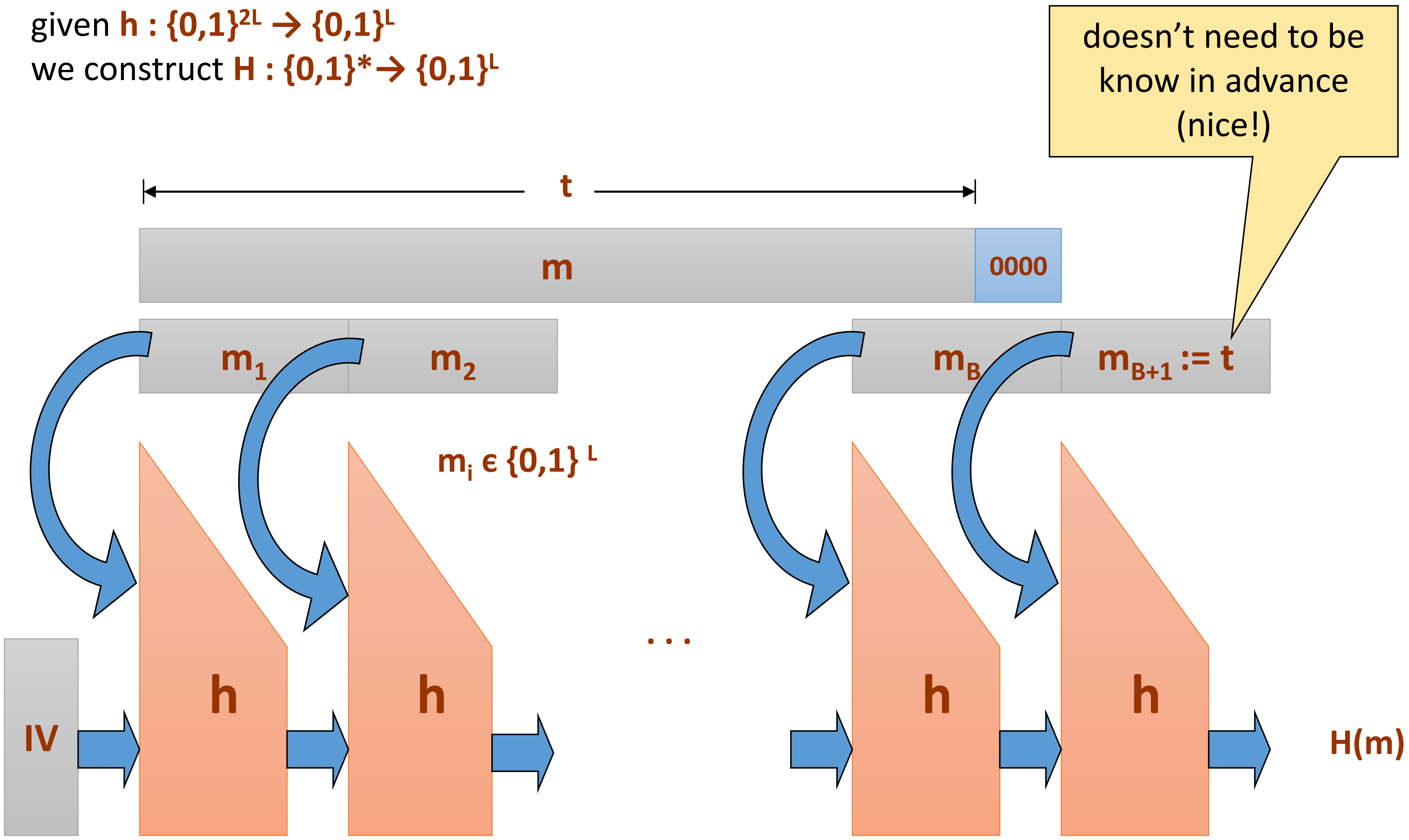
If we set **m' = m || 0000** then **H(m') = H(m).**

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.

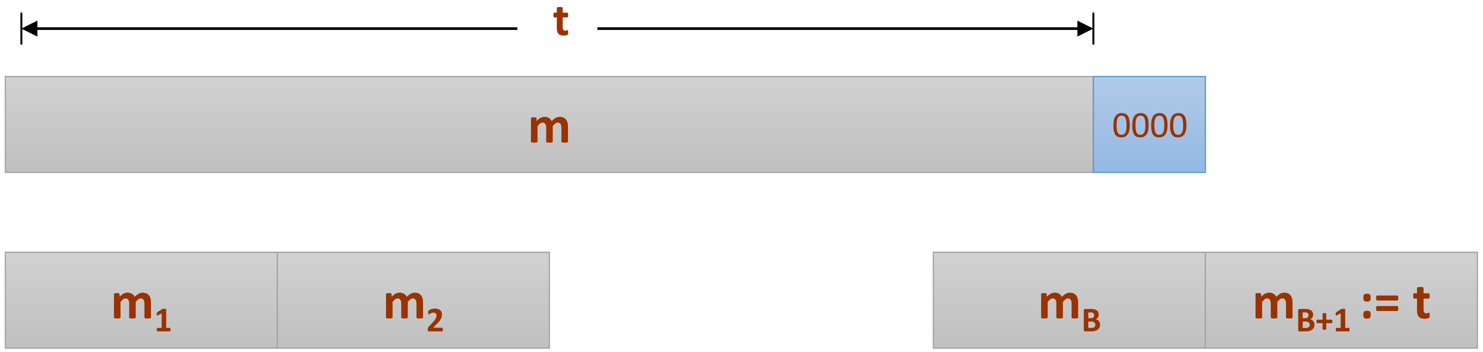
Let's prove it: How to compute a collision (x,y) in h from a collision (m,m') in H?

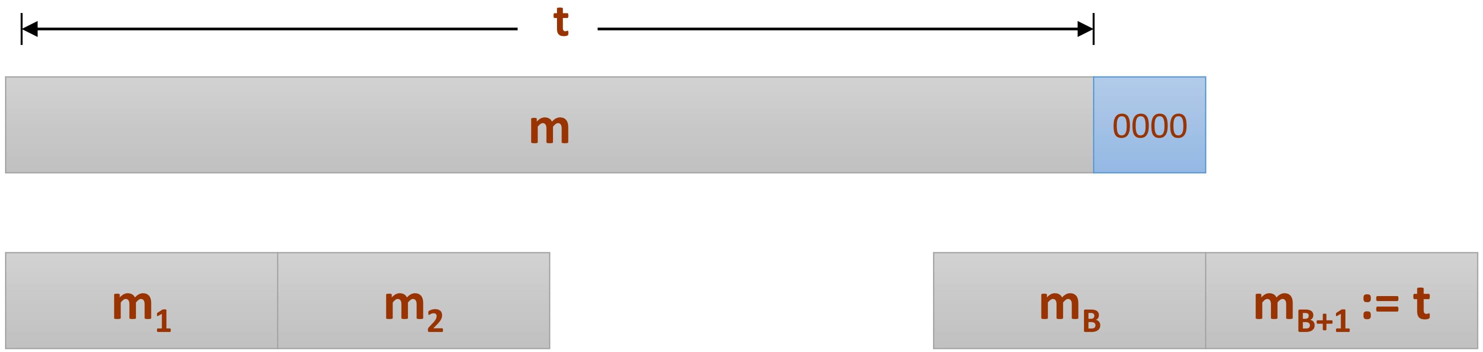
We consider two options:

1. $|m| = |m'|$

2. $|m| \neq |m'|$

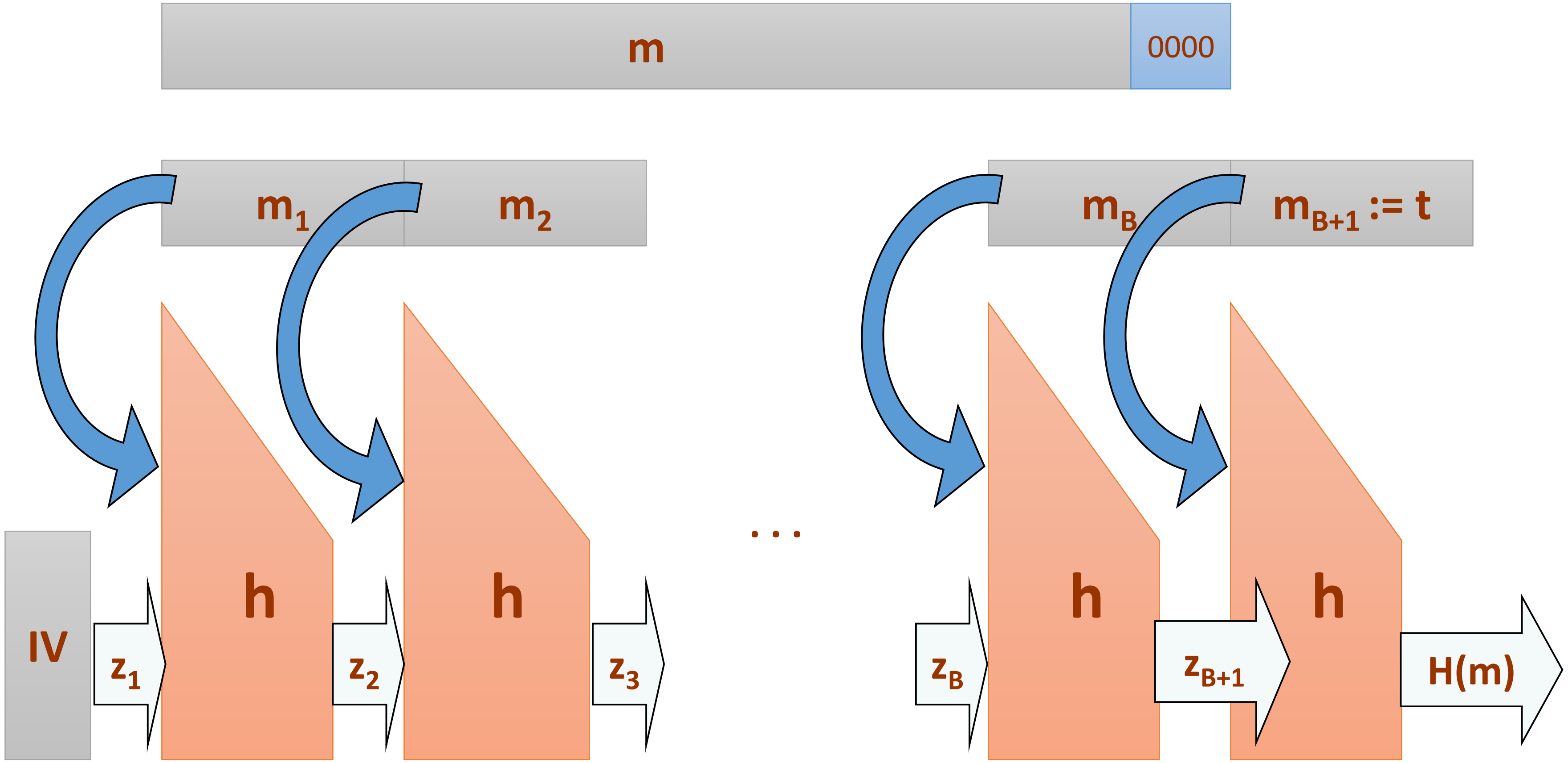
Option 1: $|m| = |m'|$





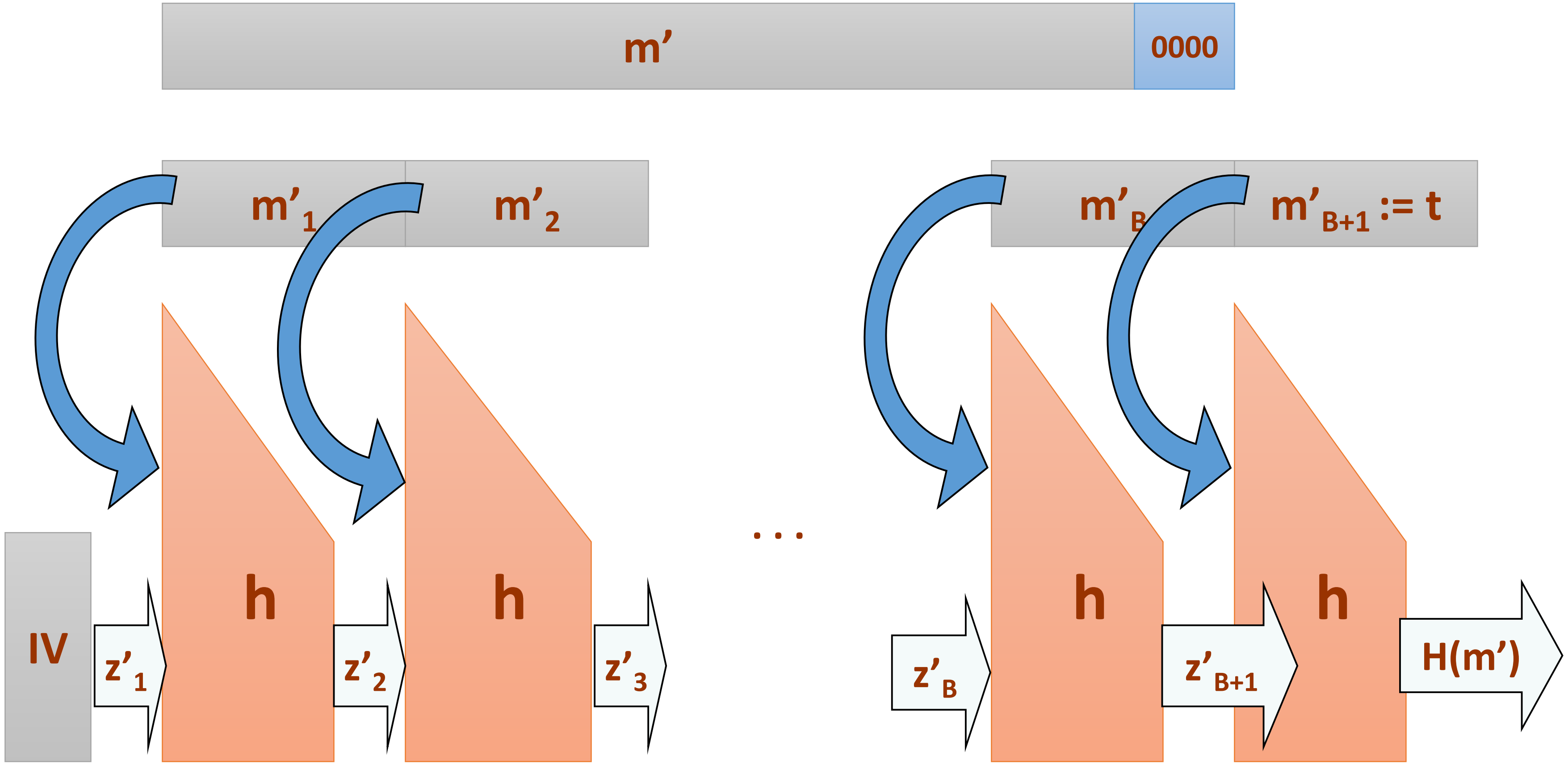
$|m| = |m'|$

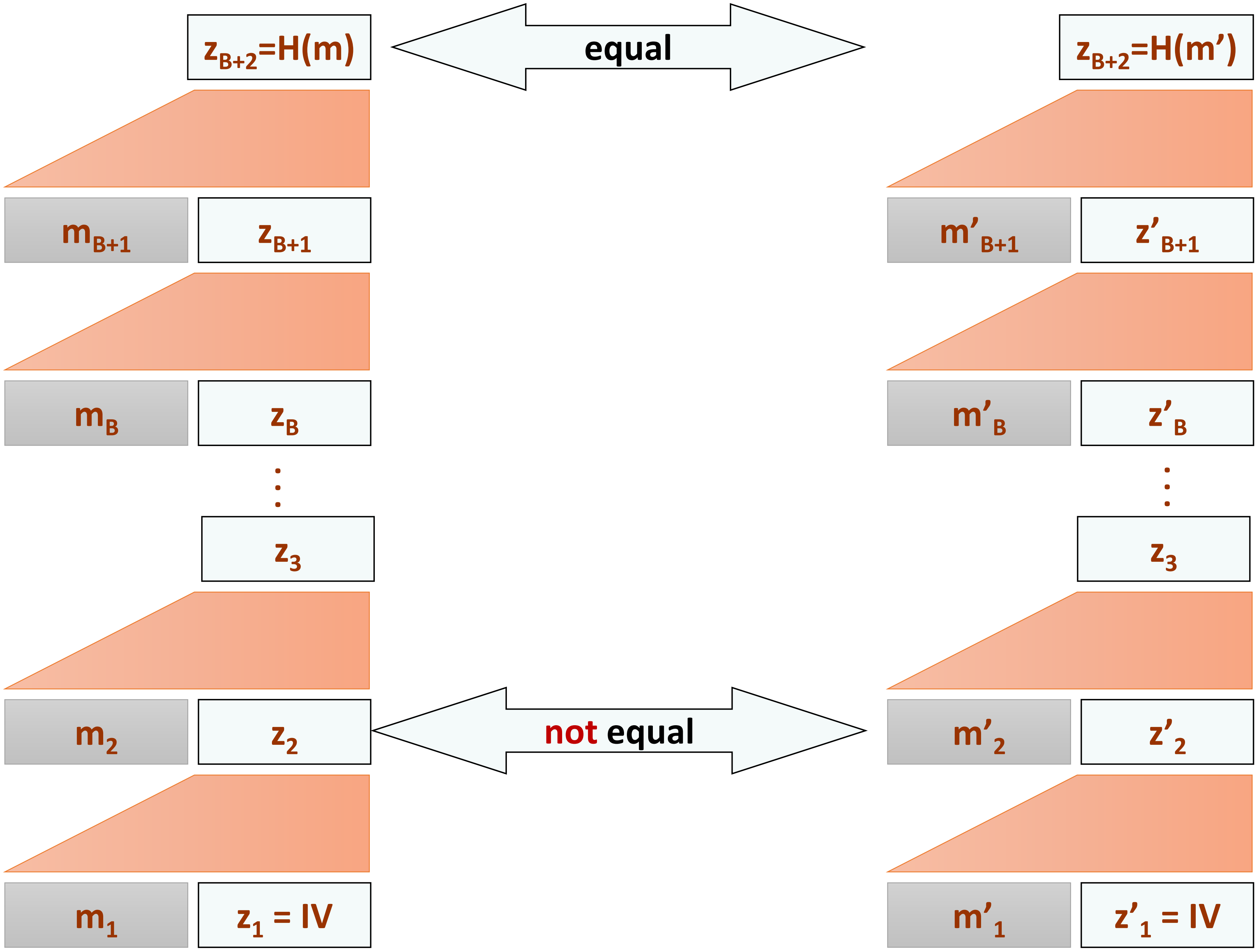
Some notation:

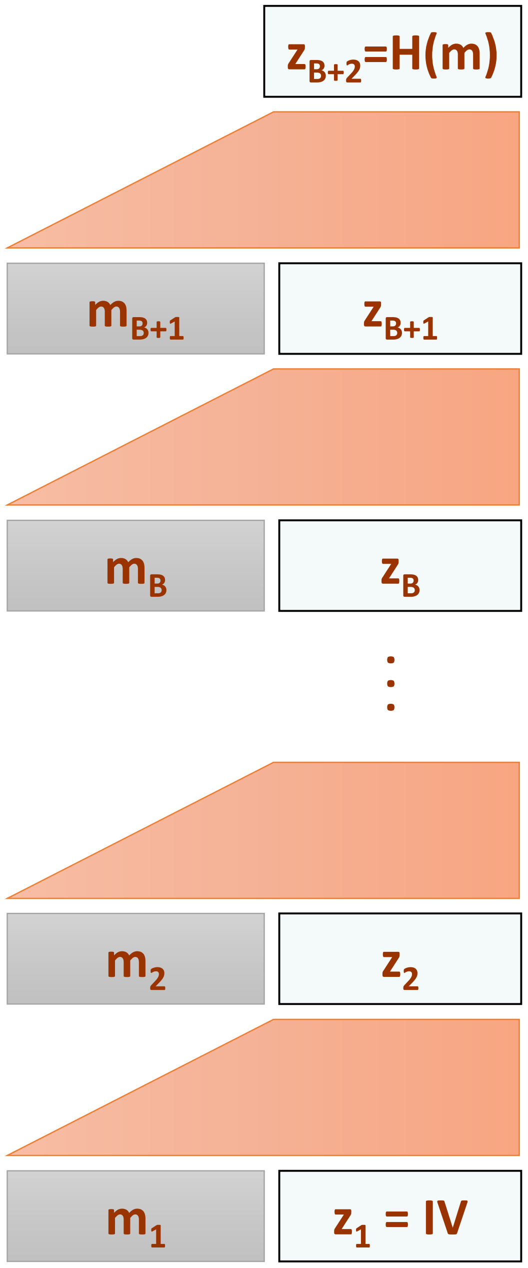


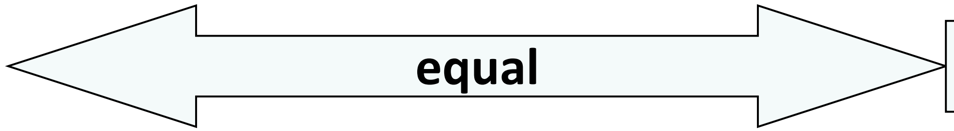
$|m| = |m'|$

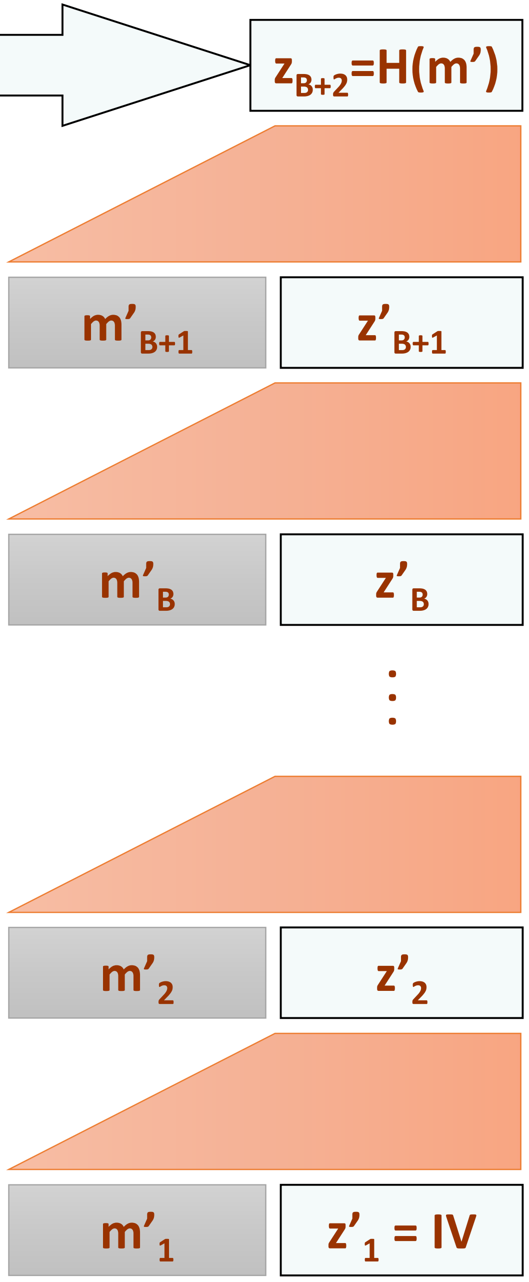
For **m'**:




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





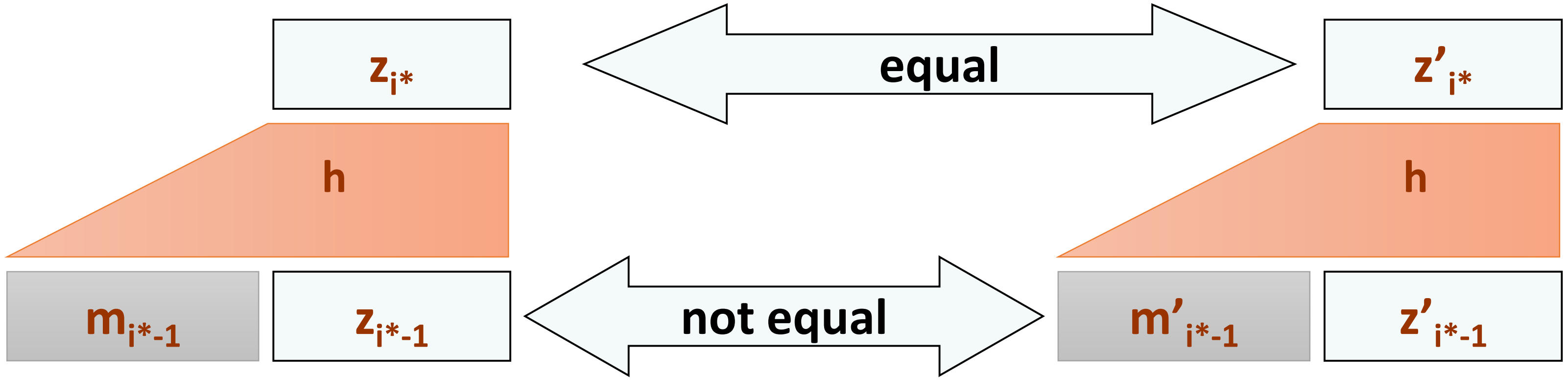


Let **i*** be the least **i** such that

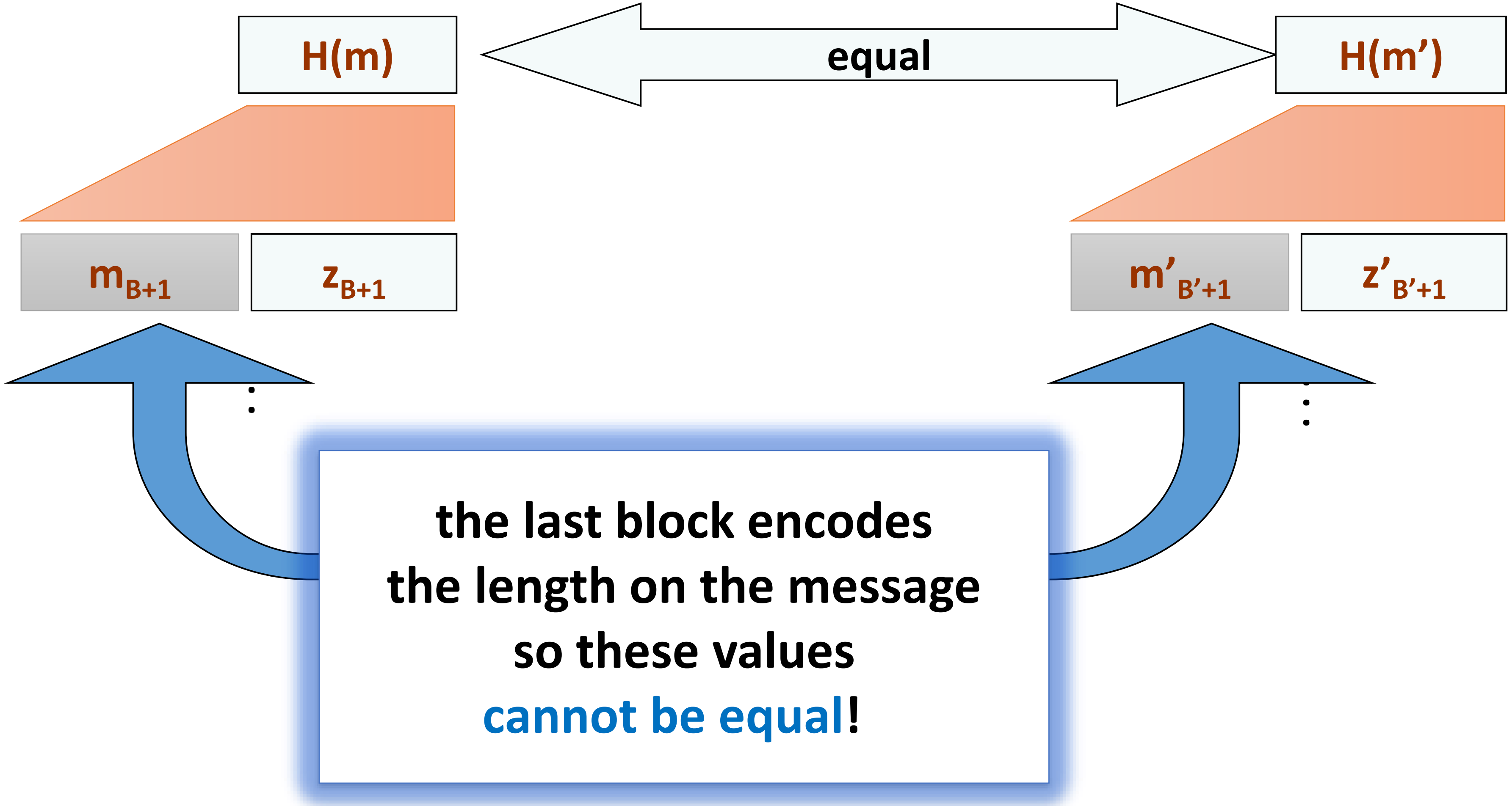
$$(m_i, z_i) = (m'_i, z'_i)$$

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

So, we have found a collision!



Option 2: $|m| \neq |m'|$



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

$$H : A \to B$$

Take **n** values

$$x_1,\dots,x_n$$

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

$$H(x_i) = H(x_j).$$

If $n \geq |B|$ 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,…
  - 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,

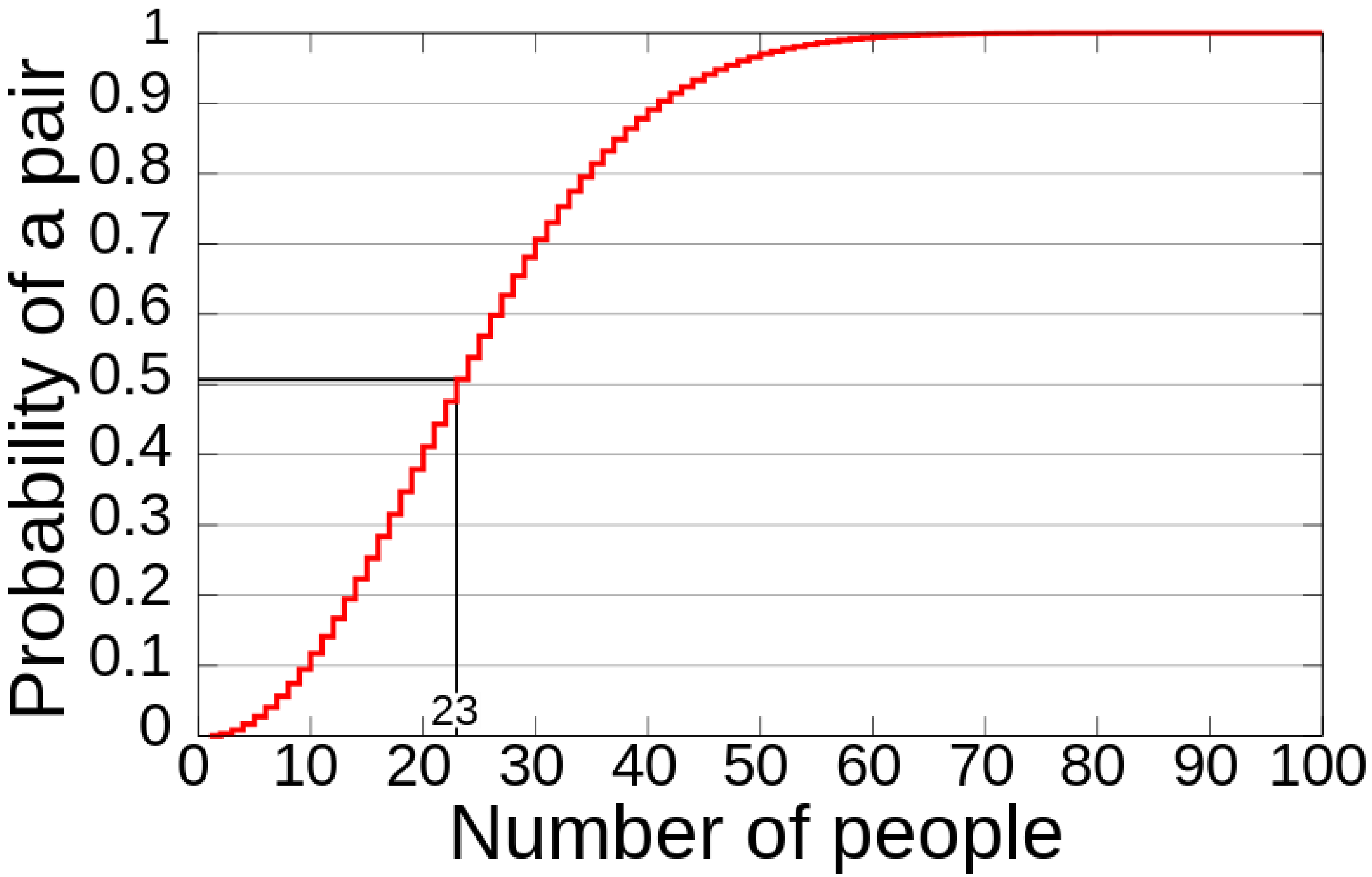
$$e^{-k(k-1)/2m} = 1 - p$$

- 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"...

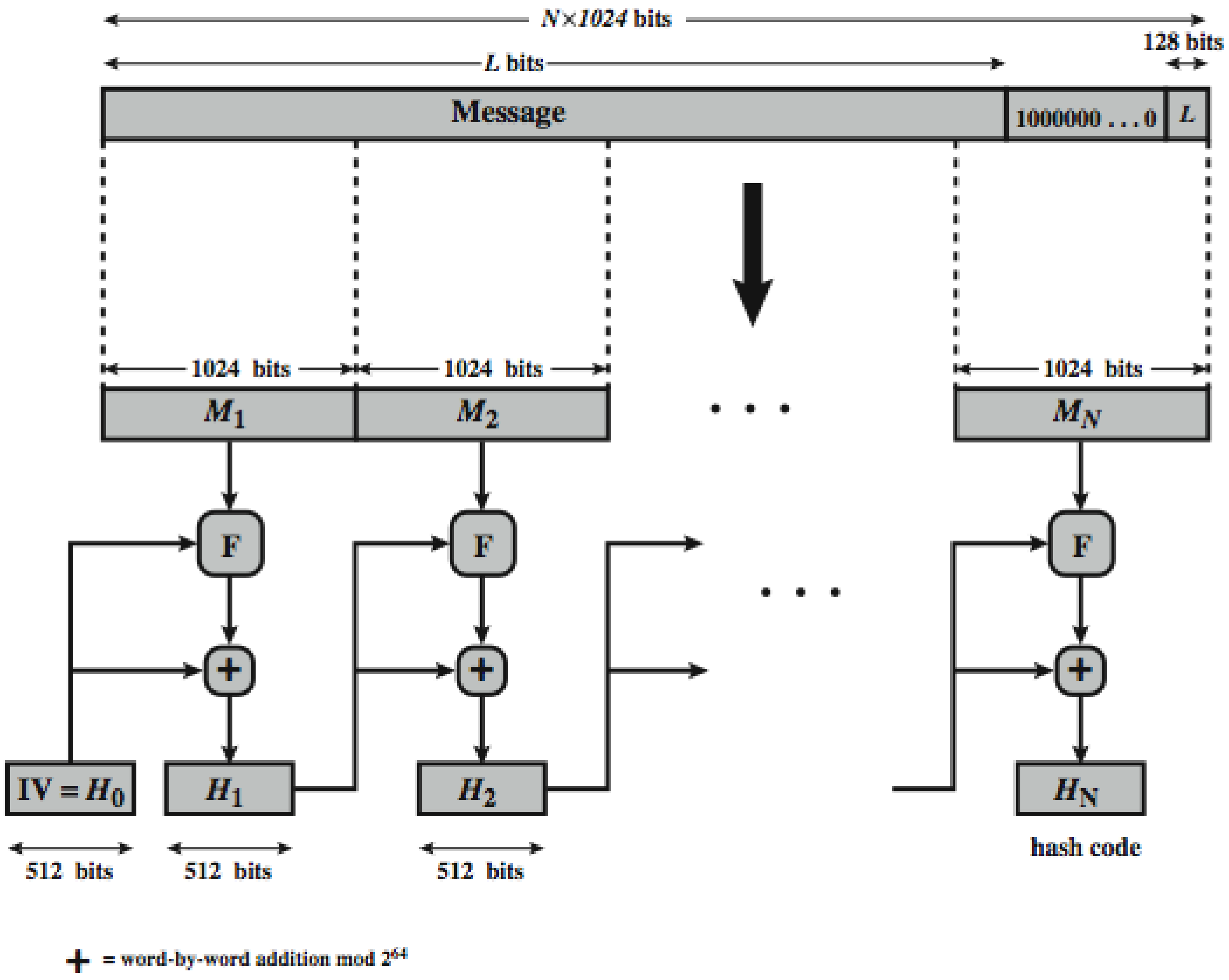
In **2005** **A. Lenstra, X. Wang, and B. de Weger** found **X.509** certificates with different public keys and the same **MD5** hash.

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.

SHA-2 overview



What the industry says about the “hash and authenticate” method?

the block cipher is still there...

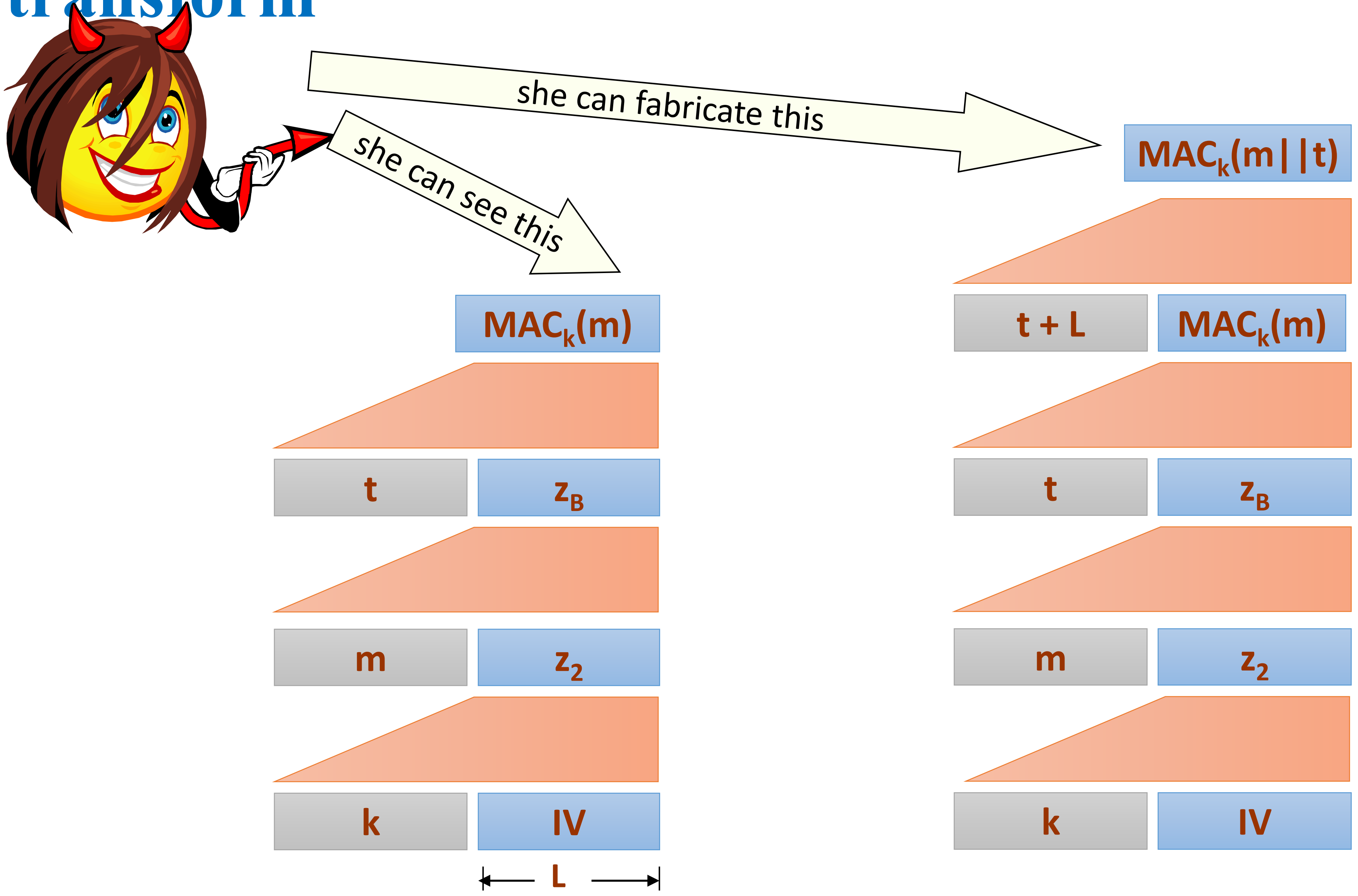
Why don’t we just hash a message together with a key:

$MAC_k(m) = H(k \,||\, m)$

?

It’s not secure!

Suppose H was constructed using the MD-transform



A better idea

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

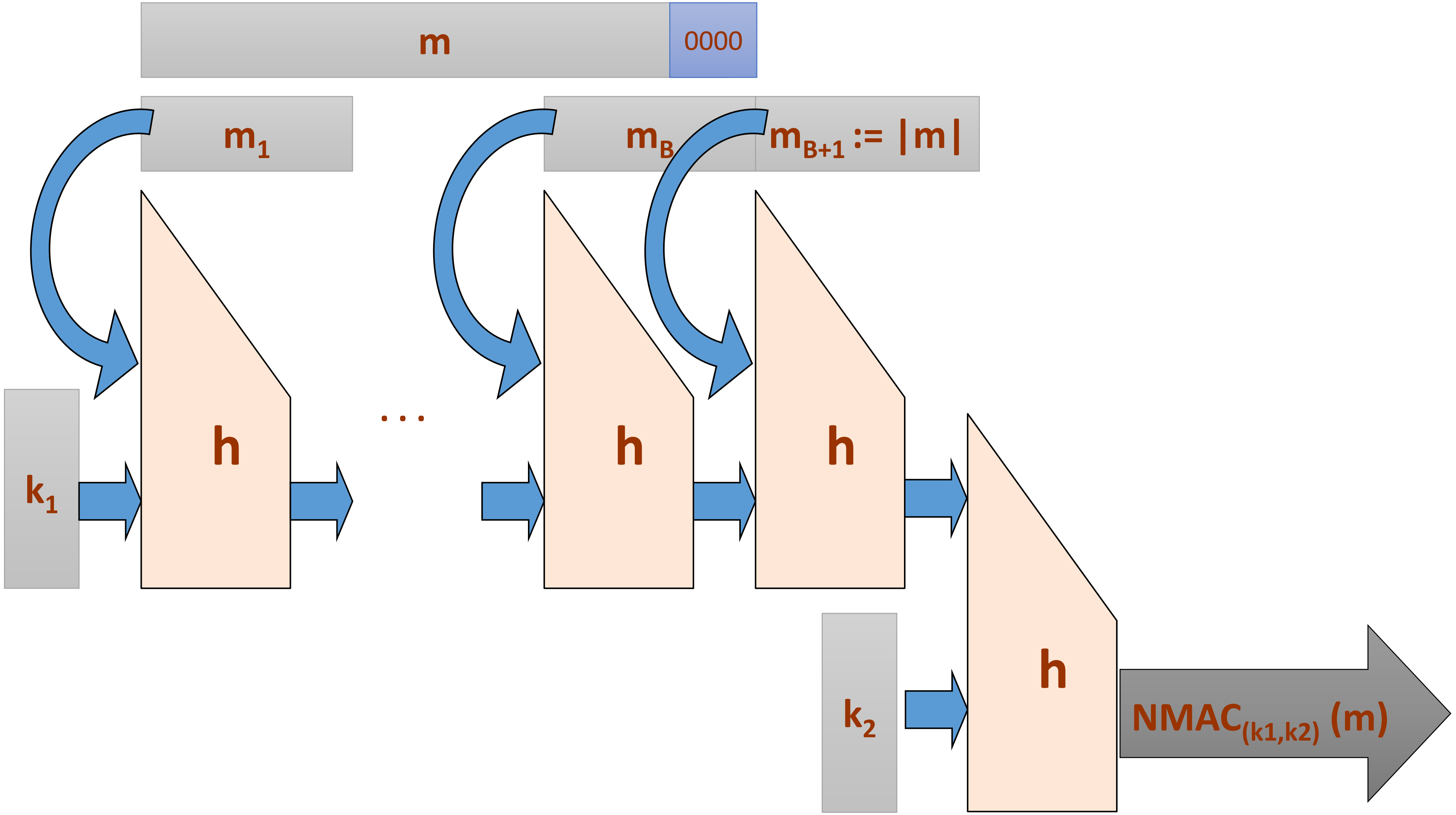
- **NMAC** (Nested MAC)
- **HMAC** (Hash based MAC)

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

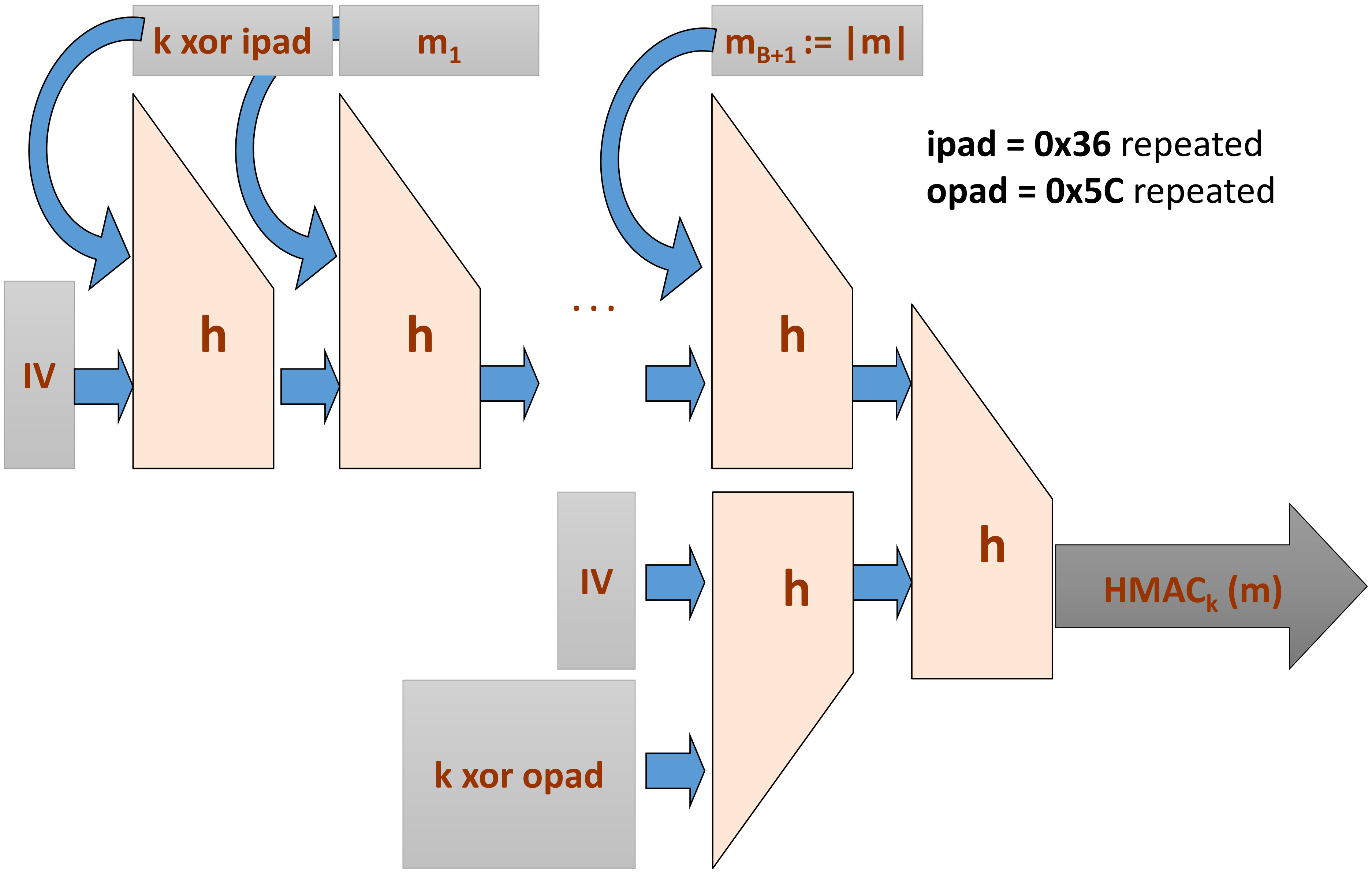


Looks better, but

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



ipad = 0x36 repeated
opad = 0x5C repeated

HMAC – the properties

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

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

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

Widely used in practice.