ENEE 457: Computer Systems Security
09/14/16

Lecture 5
Message Authentication Codes: Definition and Construction from PRPs

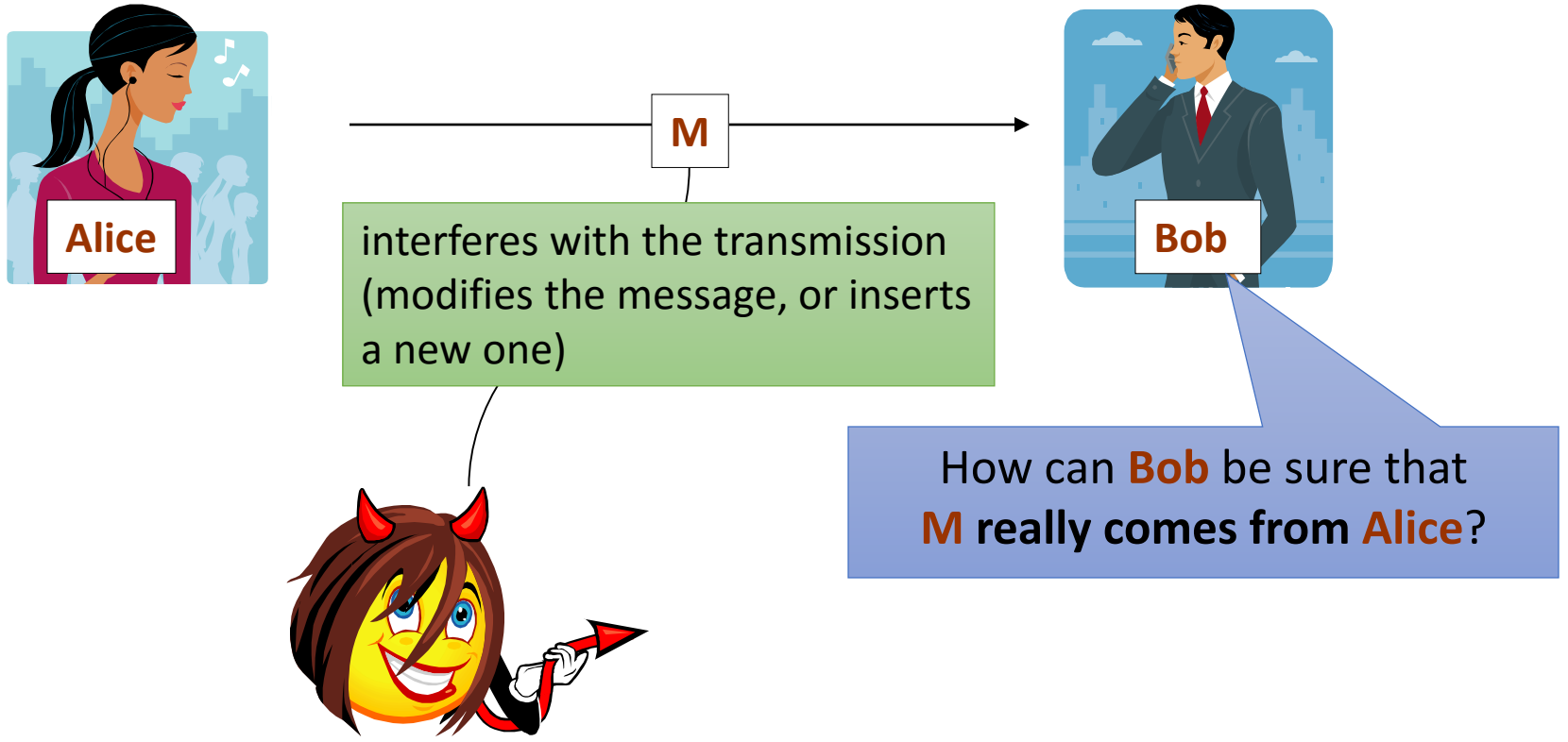
Charalampos (Babis) Papamanthou

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

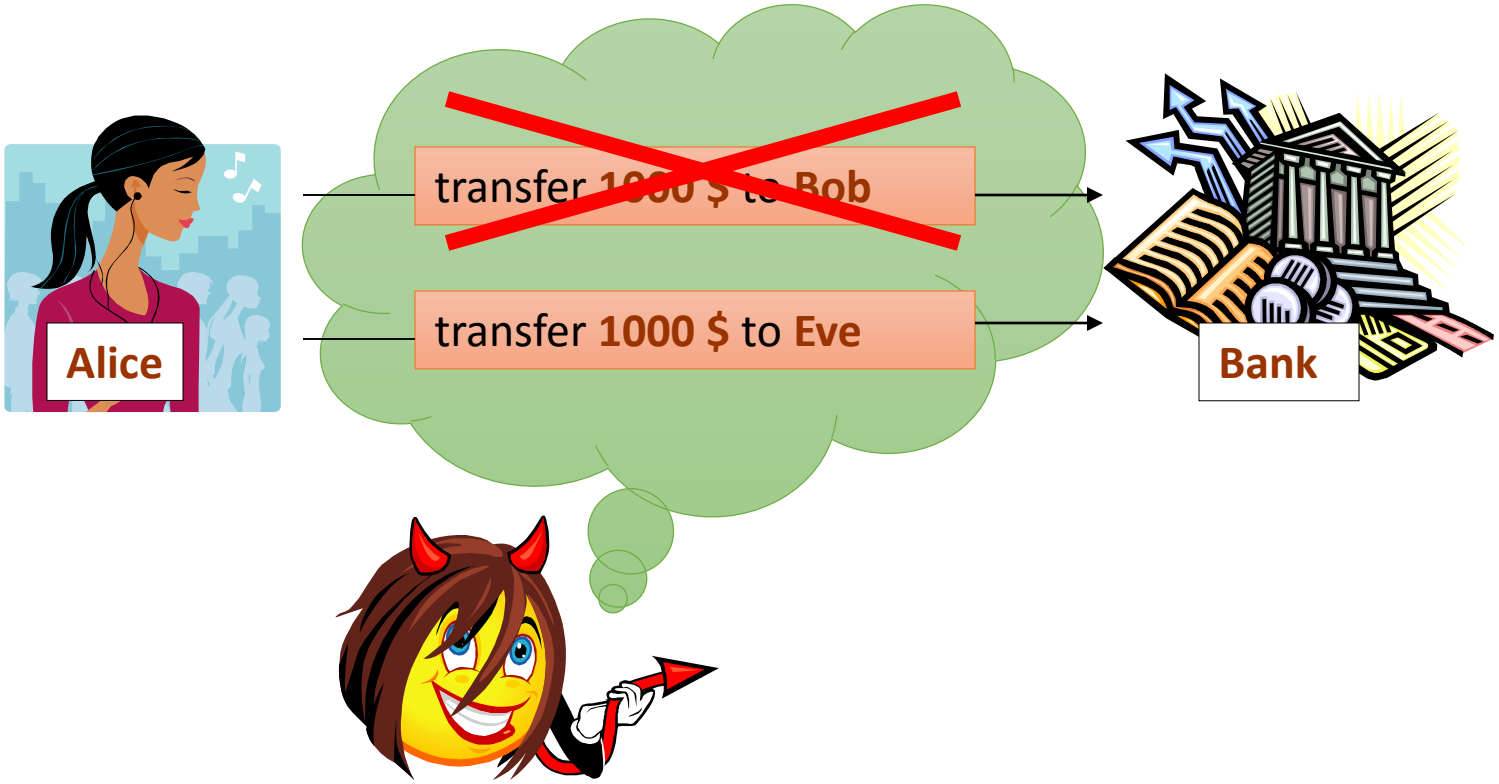
- Slides adjusted from:
  - http://dziembowski.net/Teaching/BISS09/

Message Authentication

Integrity:



Sometimes: more important than secrecy!



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

Does encryption guarantee message integrity?

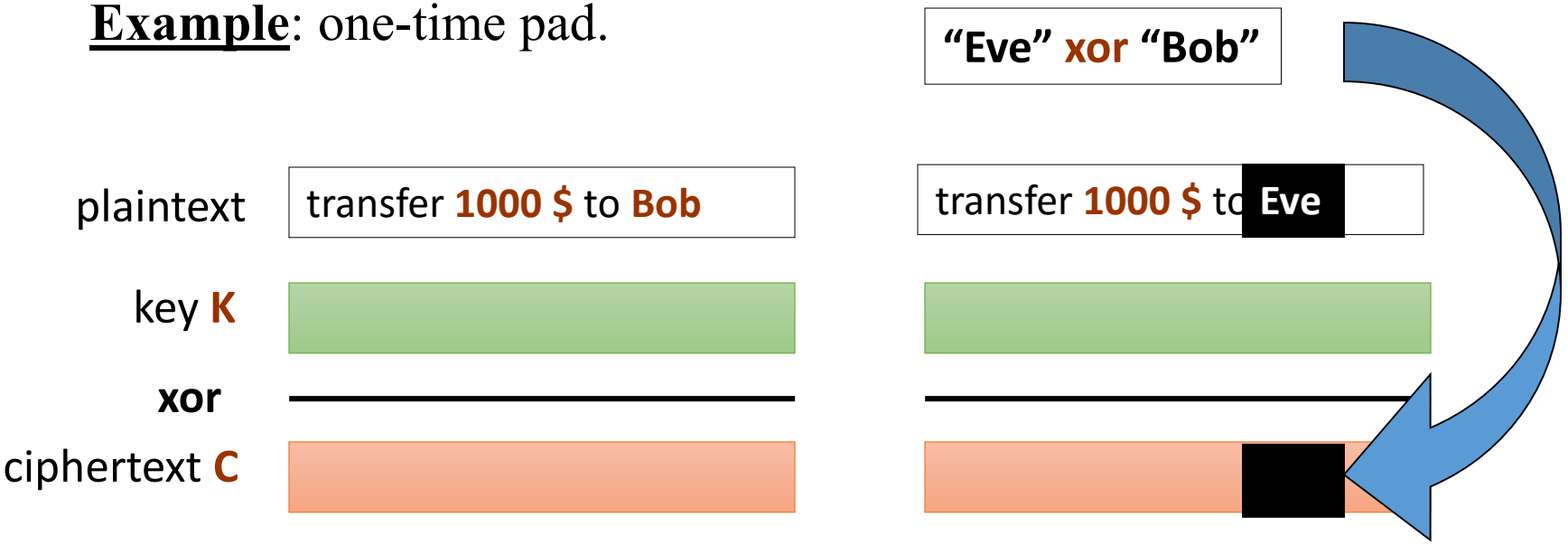
Idea:

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**.

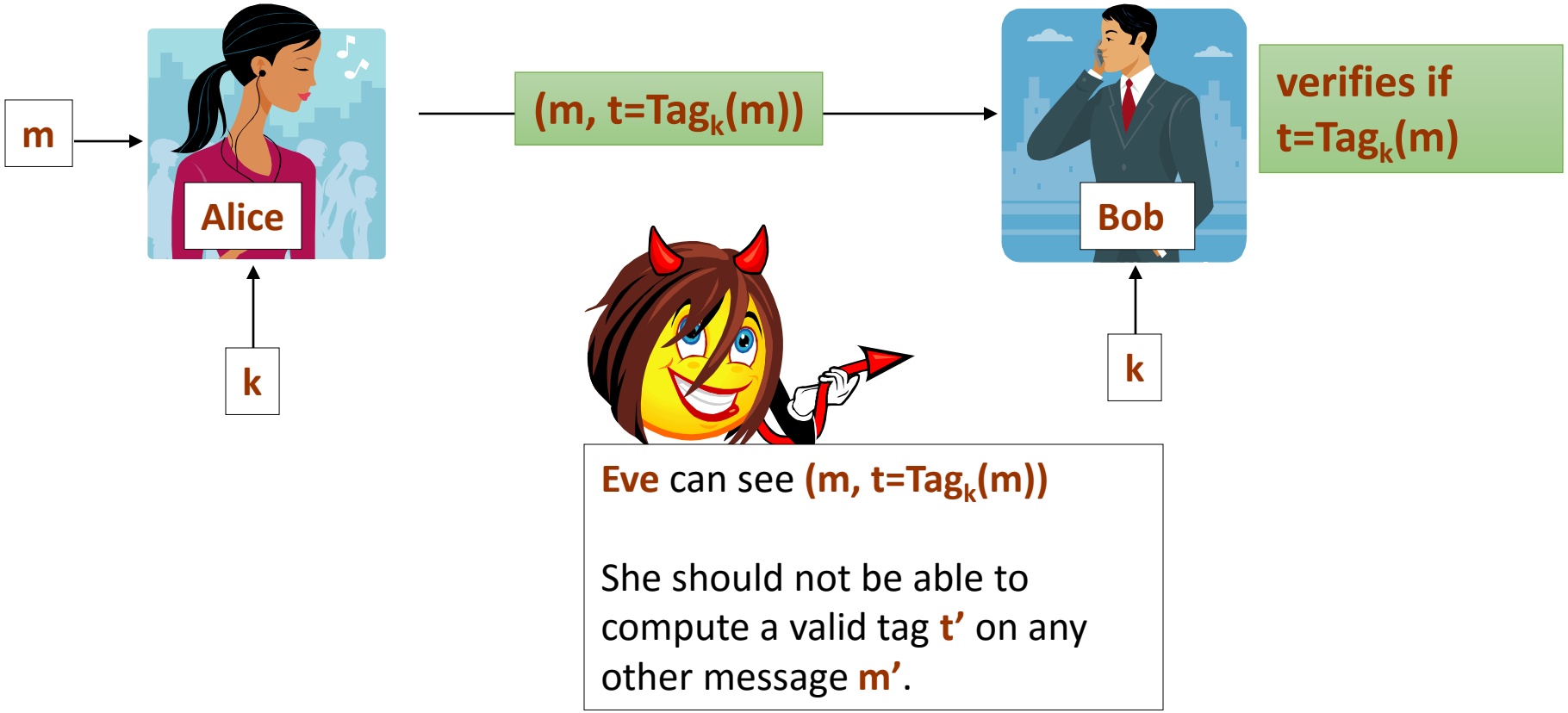
Intuiton: only **Alice** knows **k**, so nobody else can produce a valid ciphertext.

It does not work!

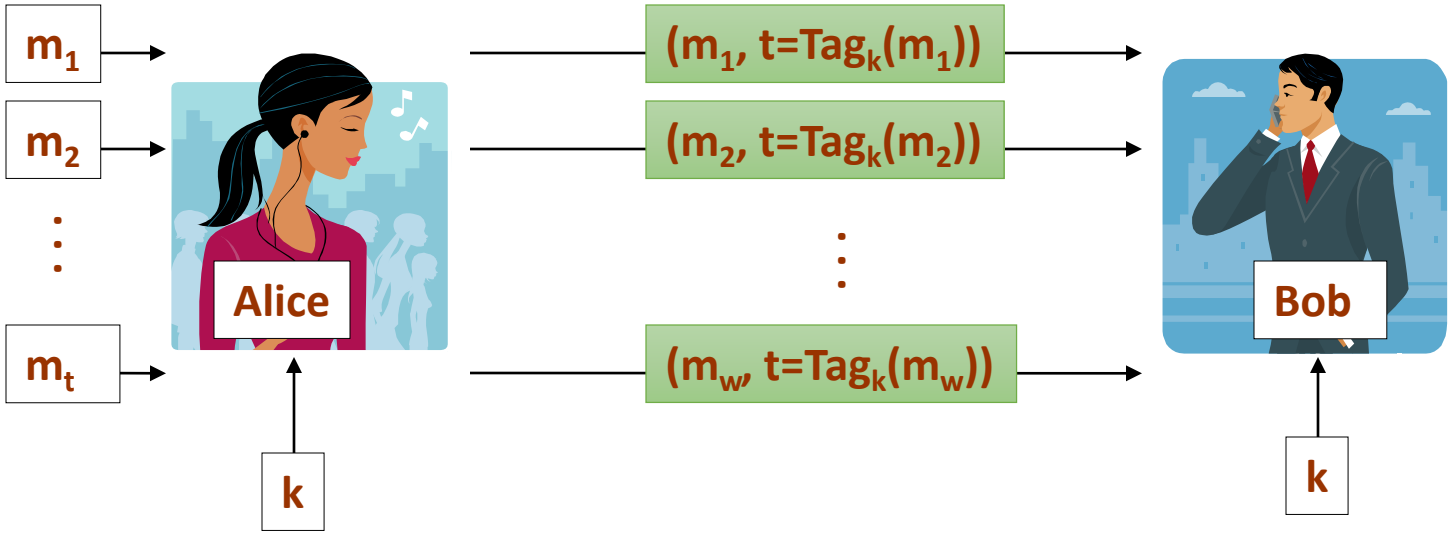
Example: one-time pad.



Message authentication

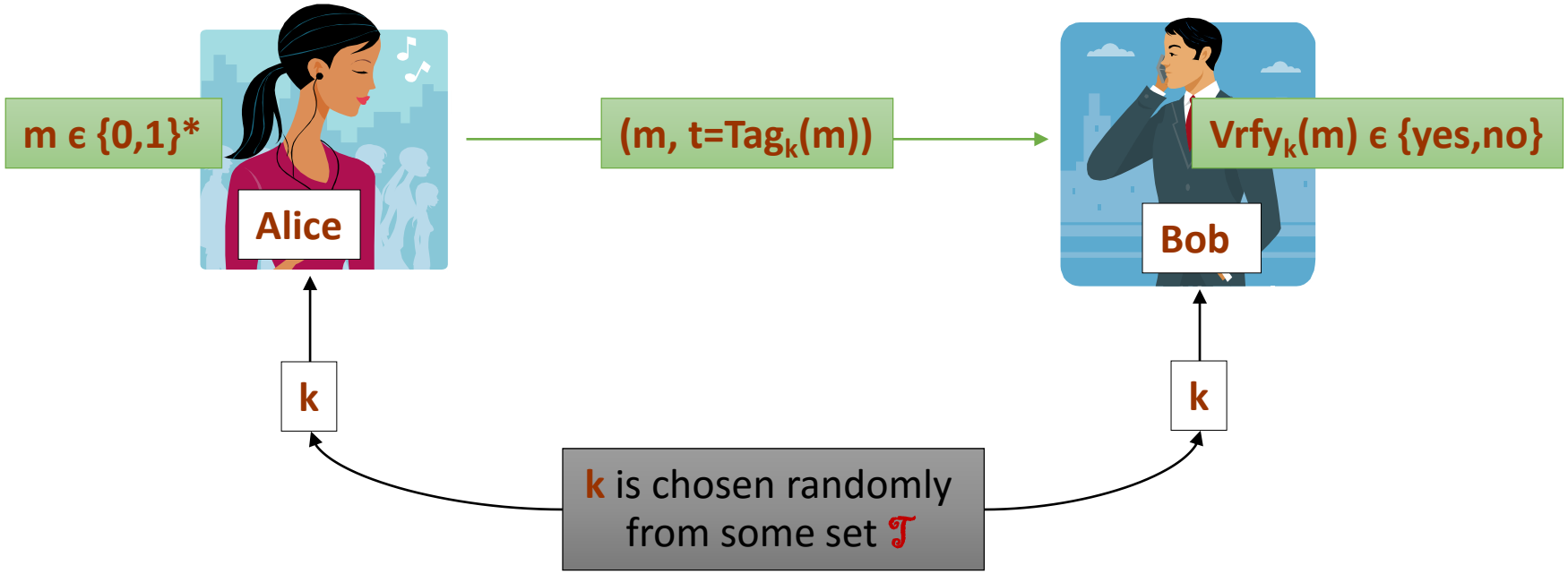


Message authentication – multiple messages



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:

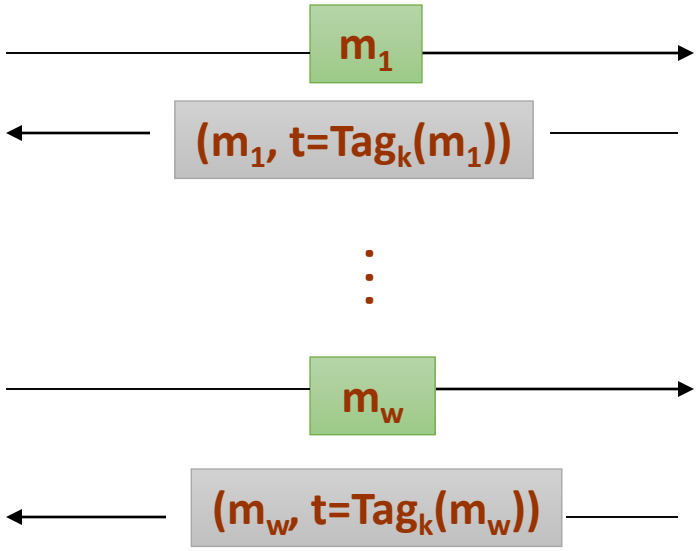
1. how the messages $m_1,\ldots,m_w$ are chosen,

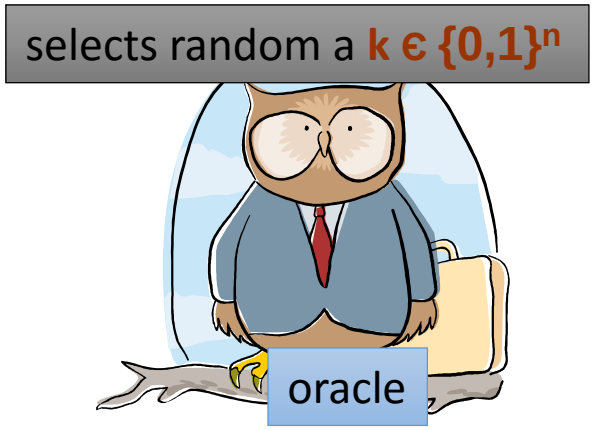
2. what is the goal of the adversary.

Good tradition: be as pessimistic as possible!

Therefore we assume that

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





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:

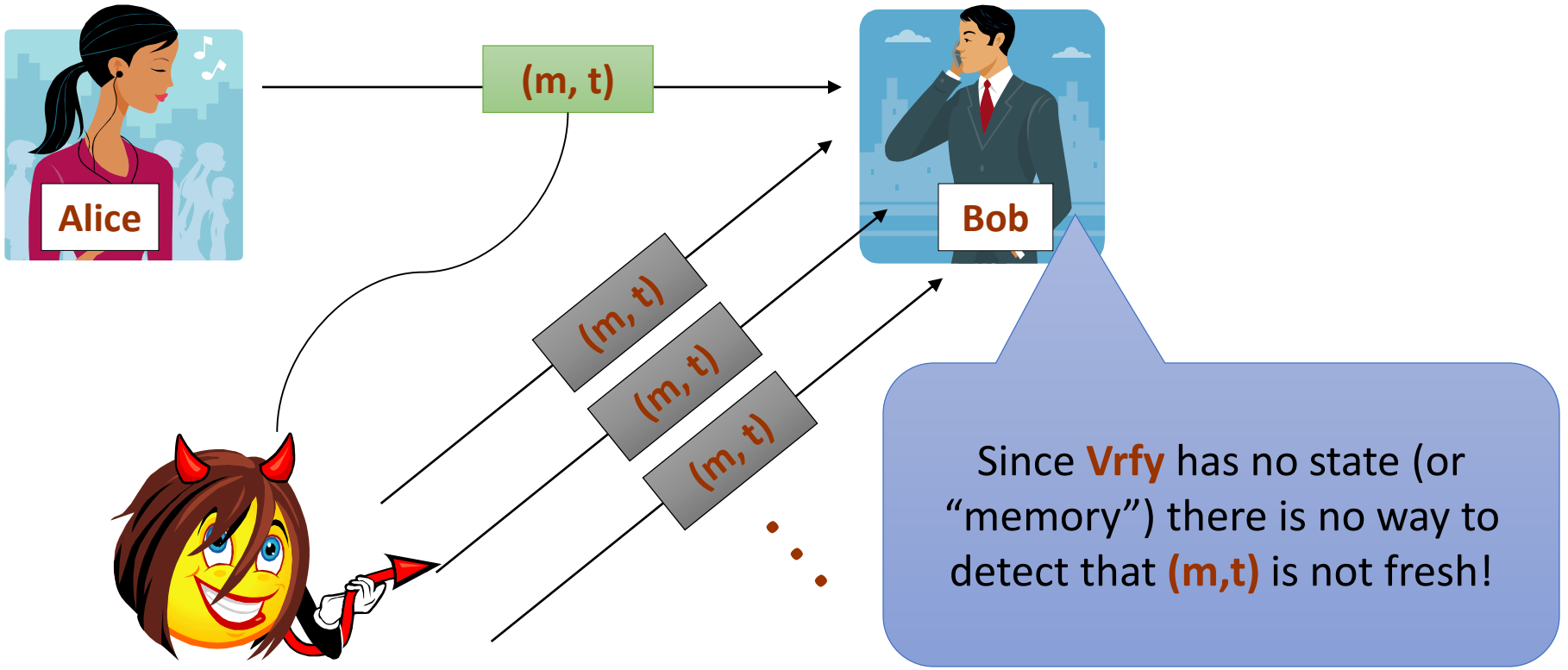
the adversary succeds only if he forges a message that “*makes sense*”.

(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:**
  $c \leftarrow Enc_{k1}(m)$ and $t \leftarrow Mac_{k2}(m)$ — wrong
- **Authenticate-then-encrypt:**
  $t \leftarrow Mac_{k2}(m)$ and $c \leftarrow Enc_{k1}(m||t)$ — better
- **Encrypt-then-authenticate:**
  $c \leftarrow Enc_{k1}(m)$ and $t \leftarrow Mac_{k2}(c)$ — the best

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

k_1 and 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

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

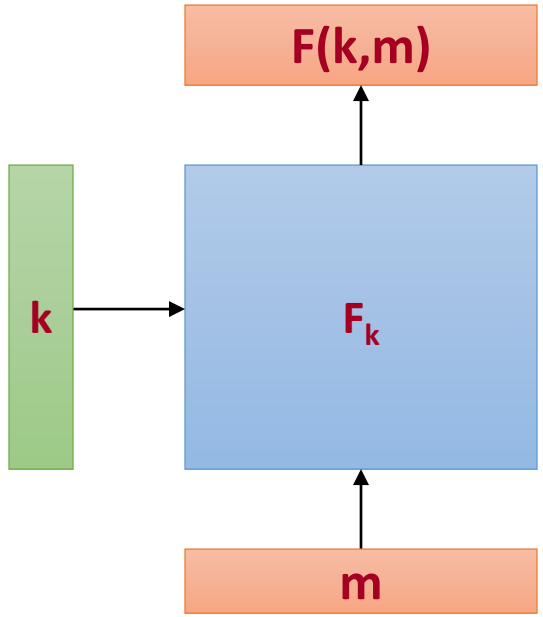
be a **block cipher**.

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

- **Mac(k,m) = F(k,m)**

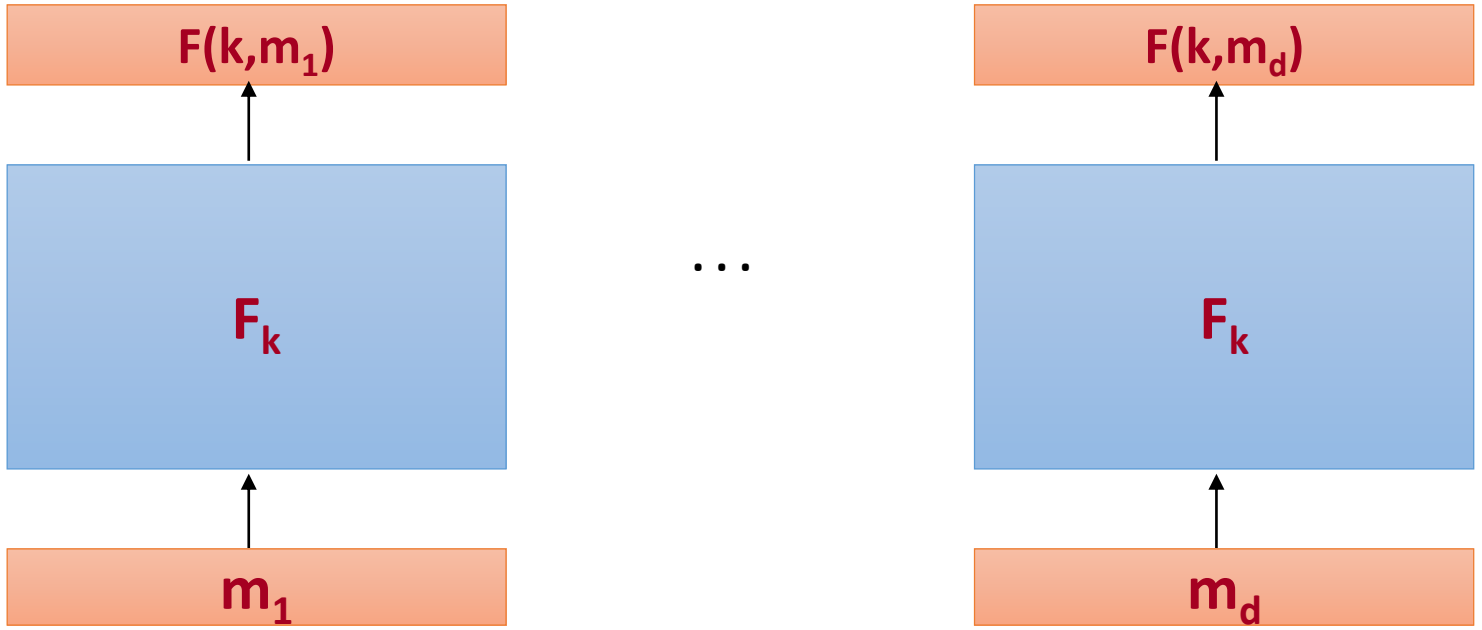
It can be proven that it is a secure **MAC**.

How to generalize it to longer messages?



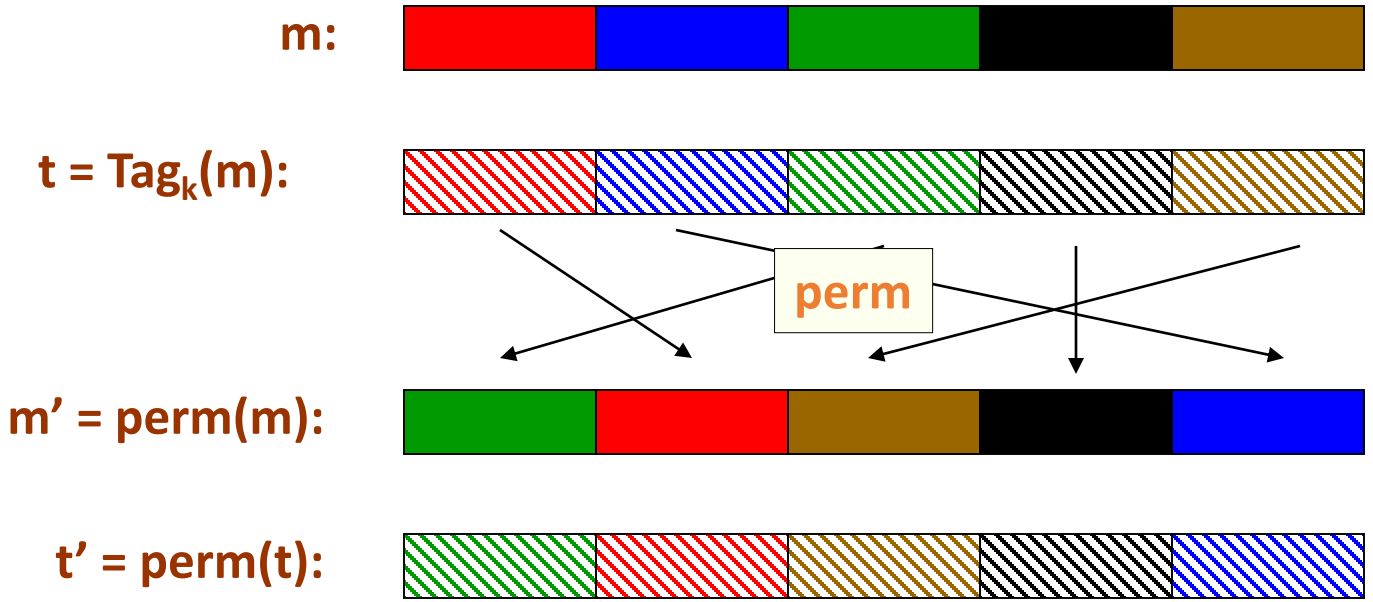
Idea 1

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



This doesn't work!

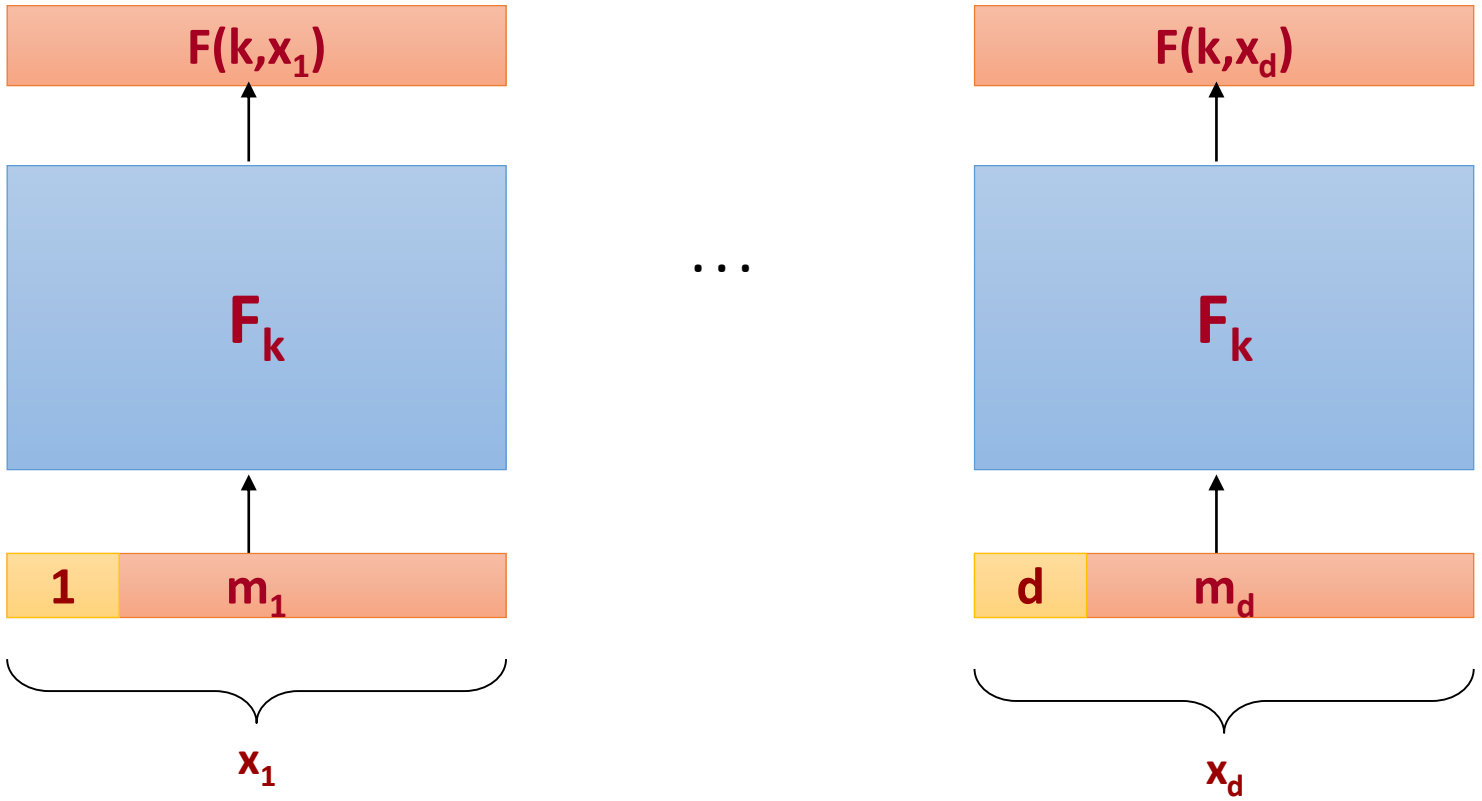
What goes wrong?



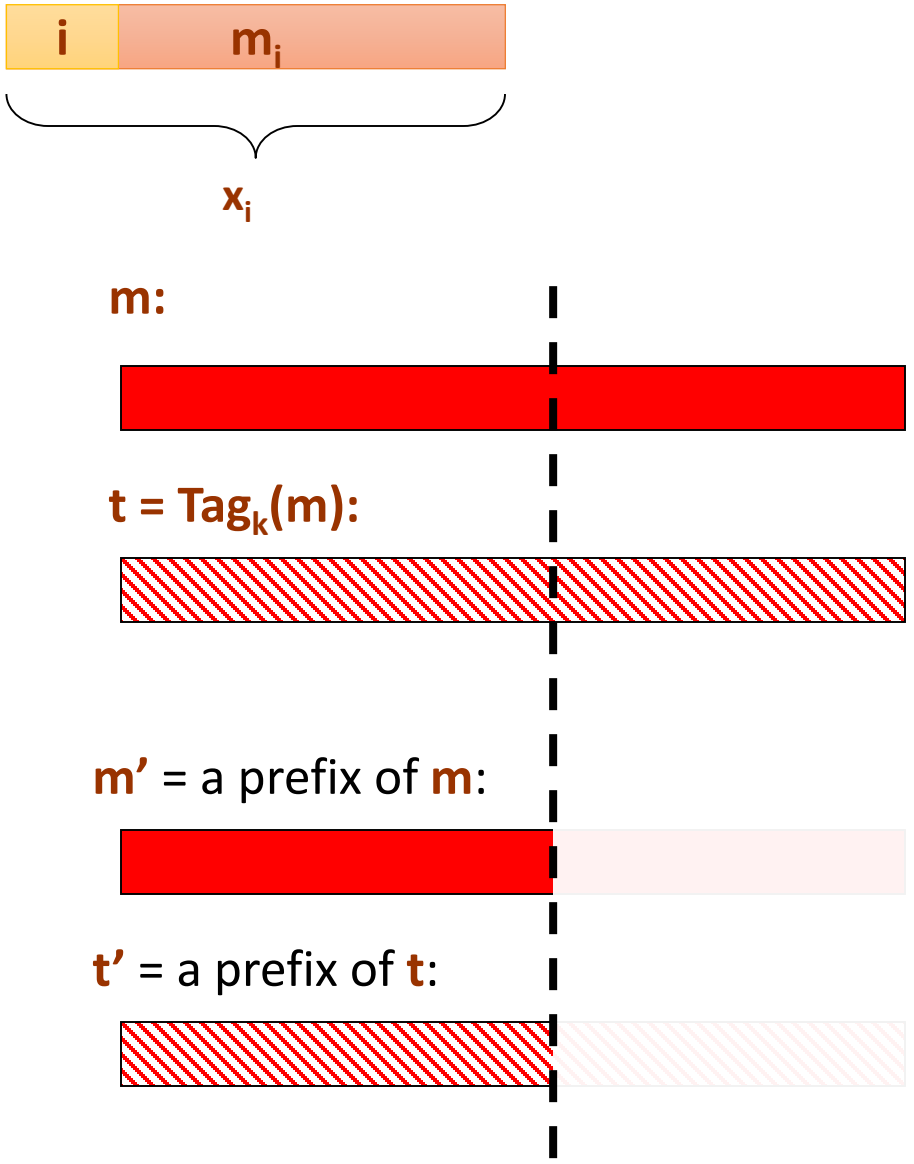
Then **t'** is a valid tag on **m'**.

Idea 2

Add a counter to each block.



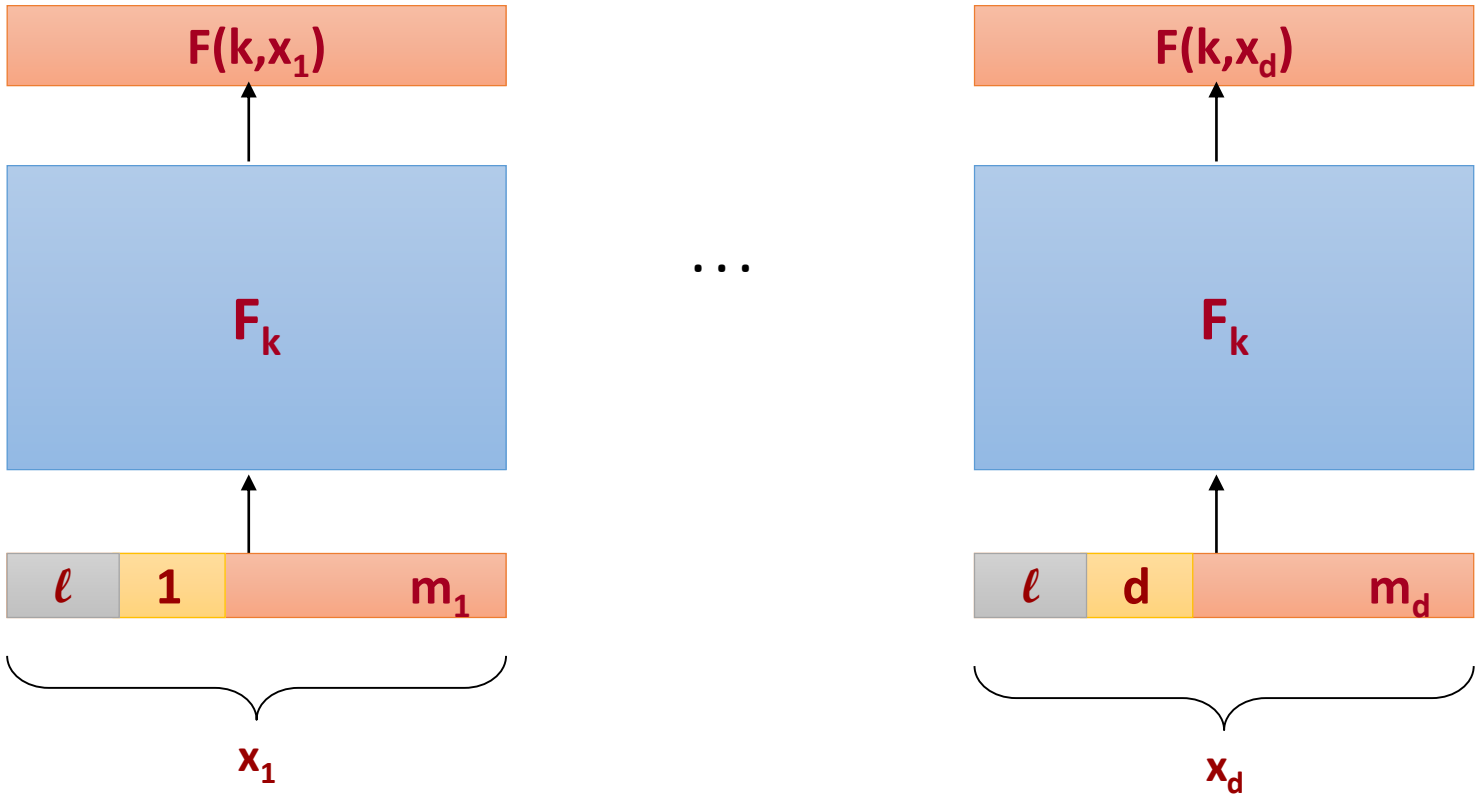
This doesn't work either!



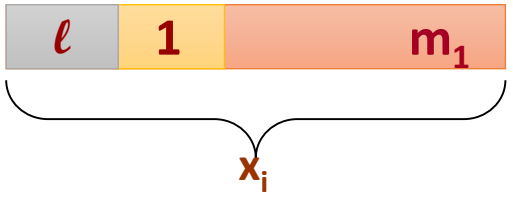
Then **t'** is a valid tag on **m'**.

Idea 3

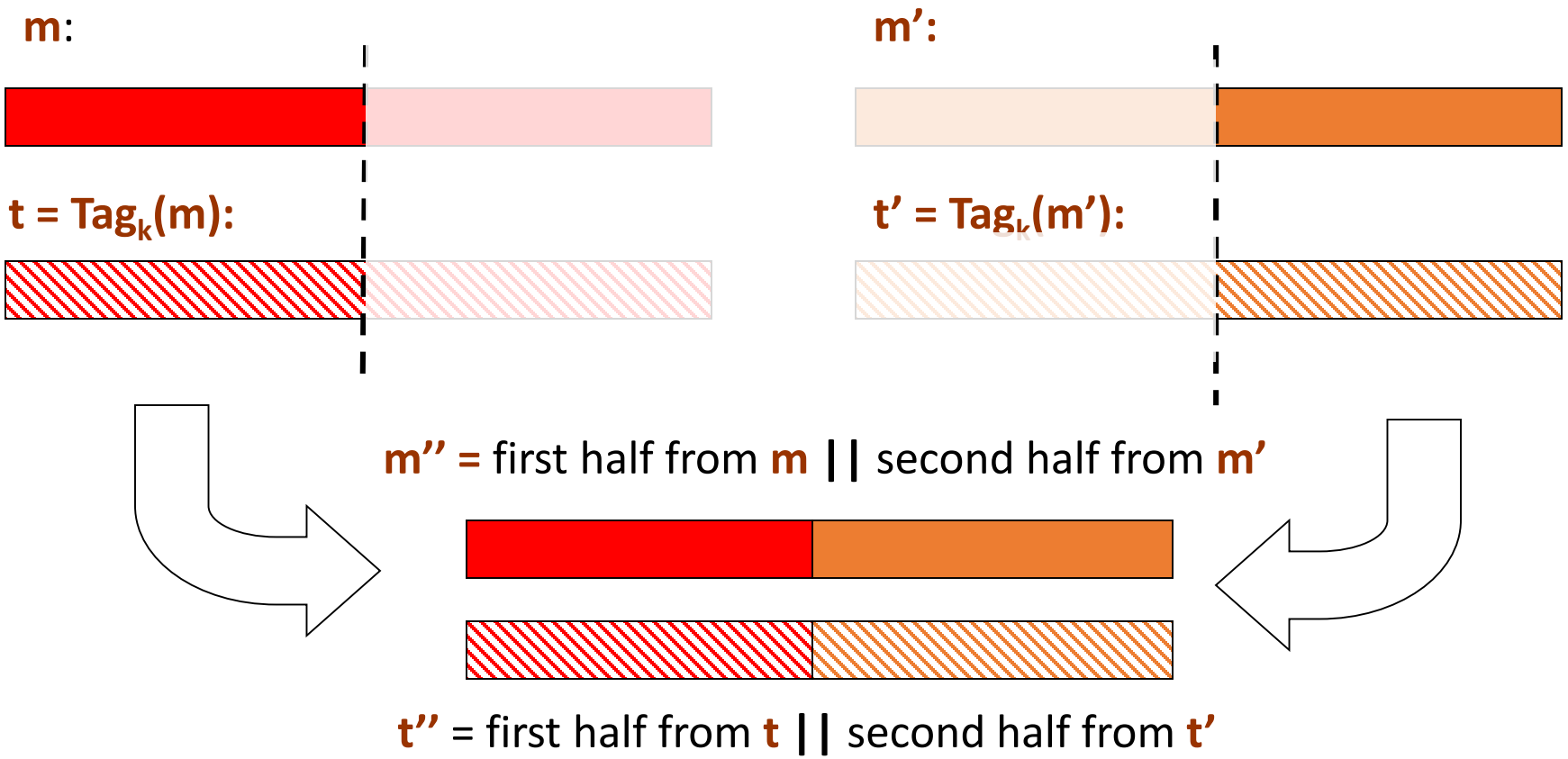
Add $\ell := |m|$ to each block



This doesn't work either!



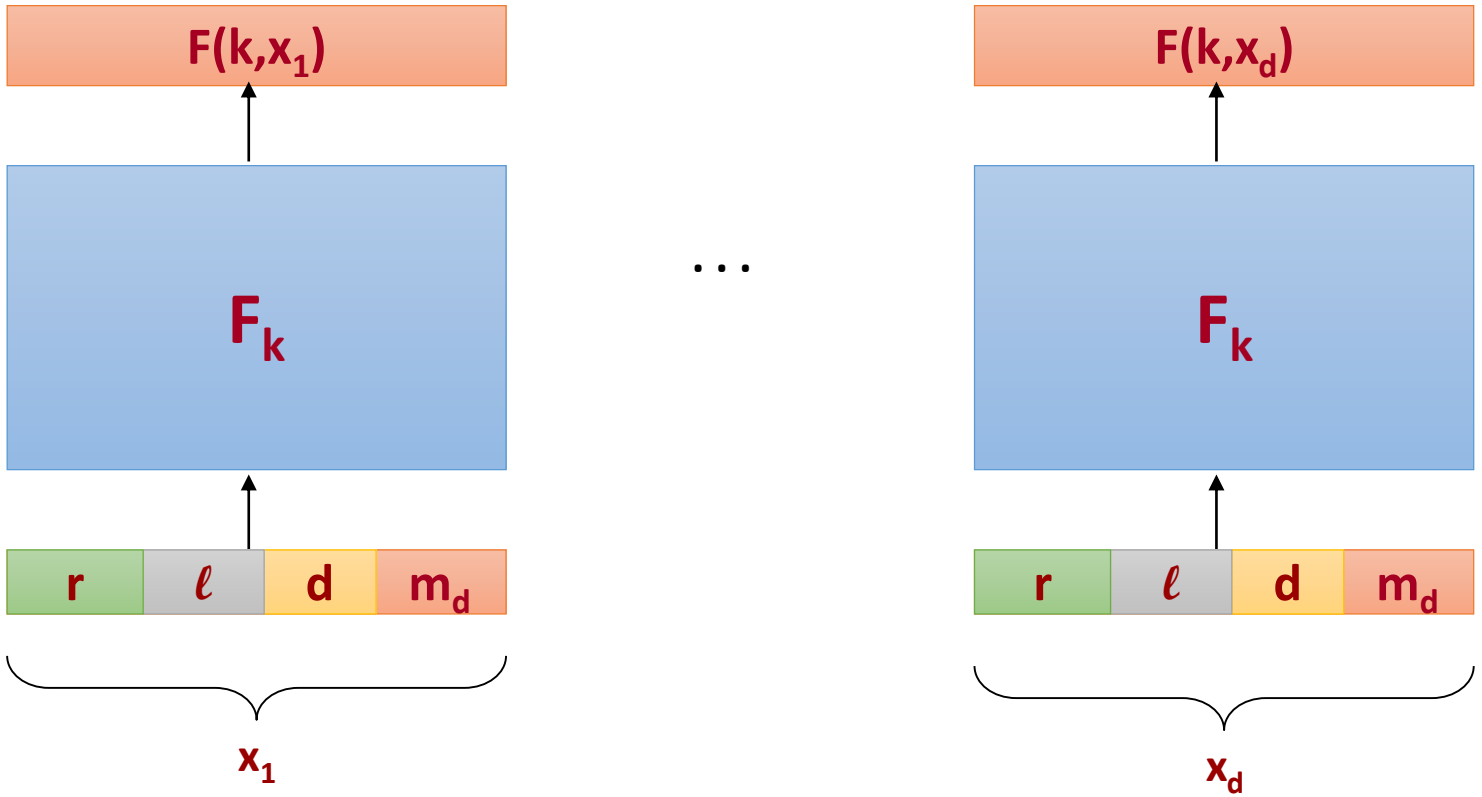
What goes wrong?



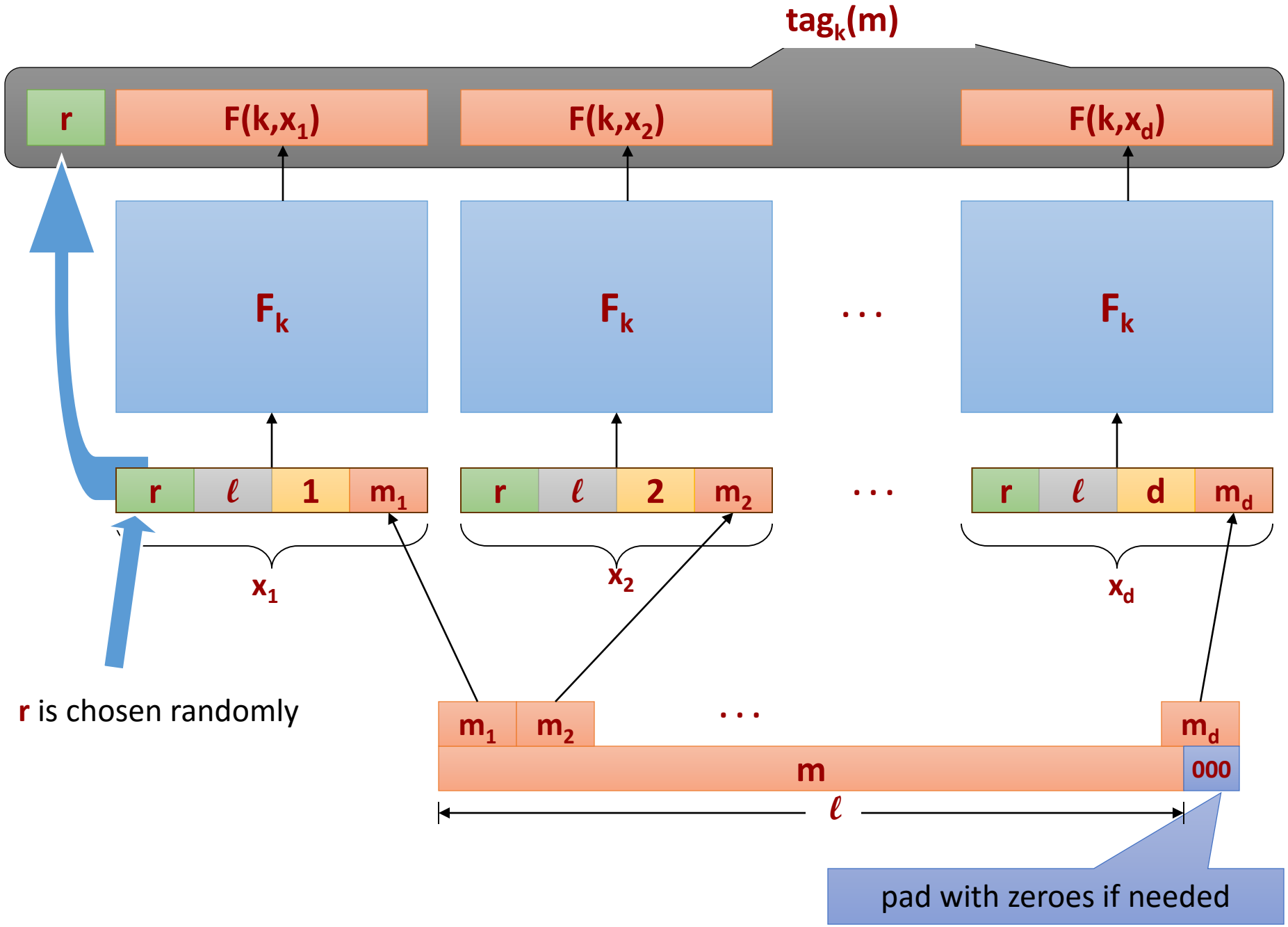
Then t'' is a valid tag on m''.

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

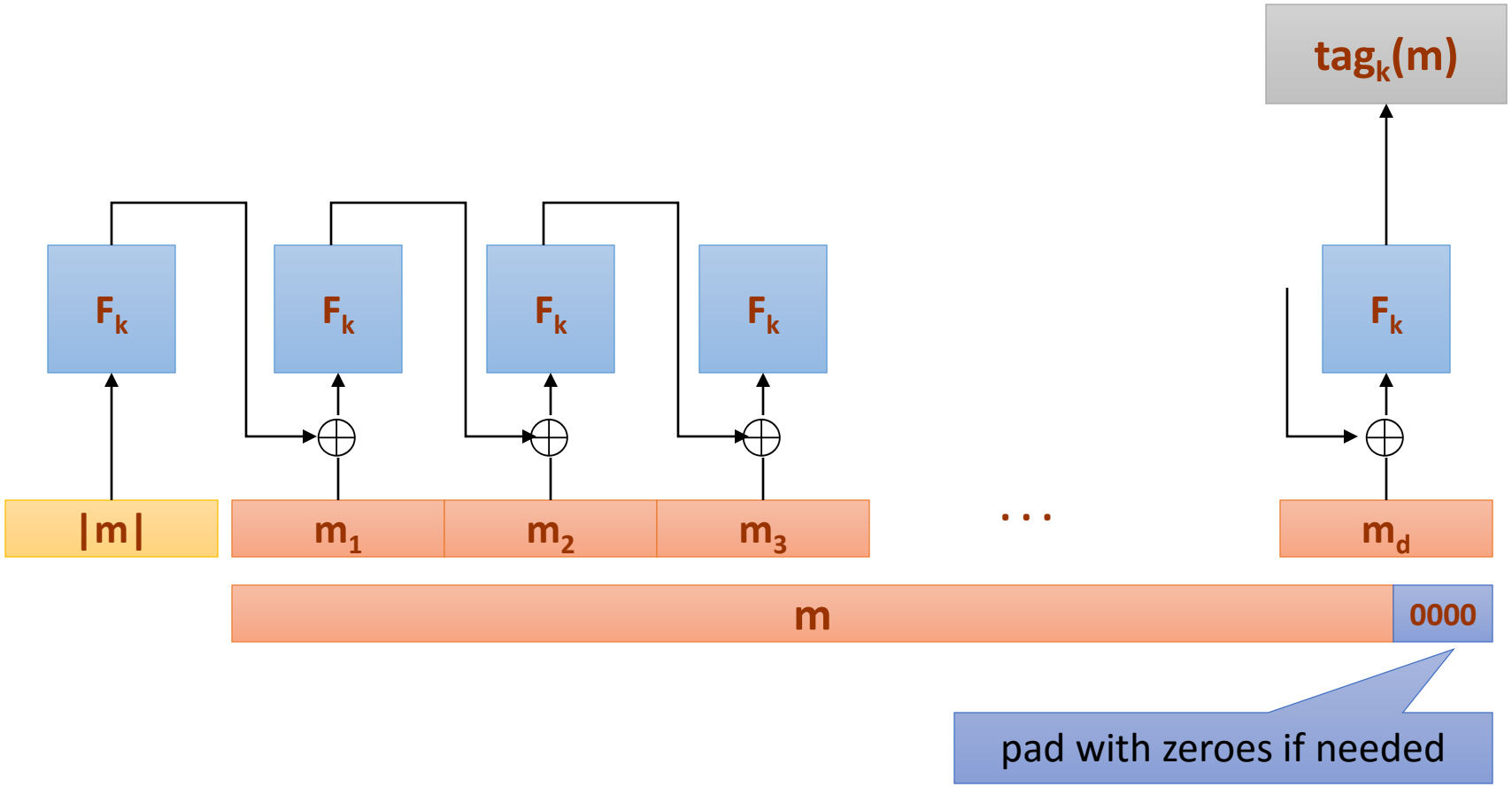
Problem:

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

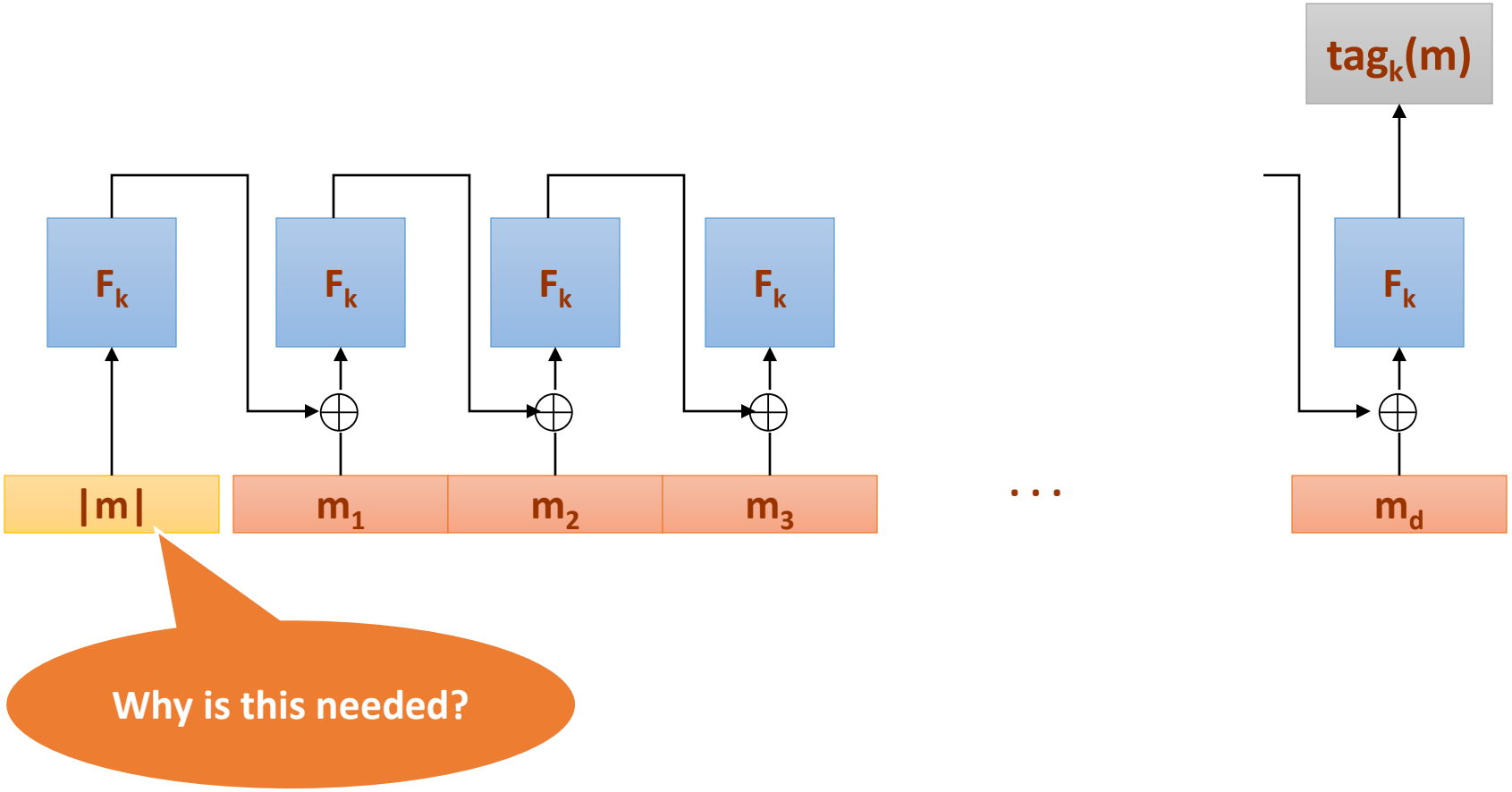
We can do much better!

CBC-MAC

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

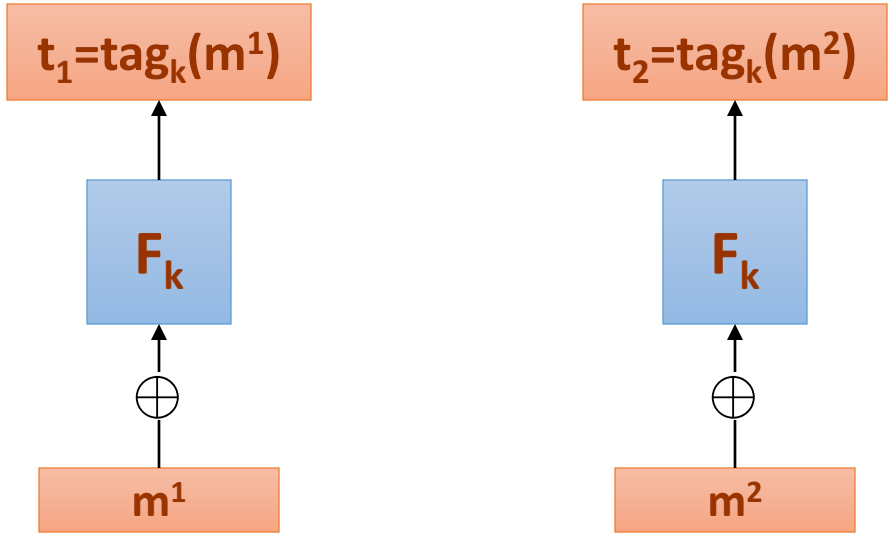


Other variants exist!



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

the adversary chooses:



now she can compute: