Lecture 5b
Message Authentication

Stefan Dziembowski

www.crypto.edu.pl/Dziembowski

University of Warsaw

31.10.18 **version 1.0**

Secure communication

	encryption	**authentication**
private key	**1** private key encryption	**2** private key authentication
public key	**3** public key encryption	**4** signatures

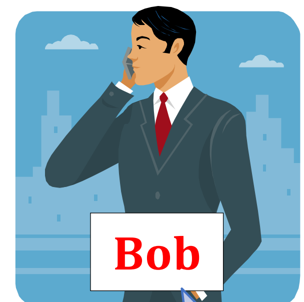
Plan

1. Introduction to Message Authentication Codes (MACs).
2. Constructions of MACs from block ciphers

Message Authentication

Integrity:

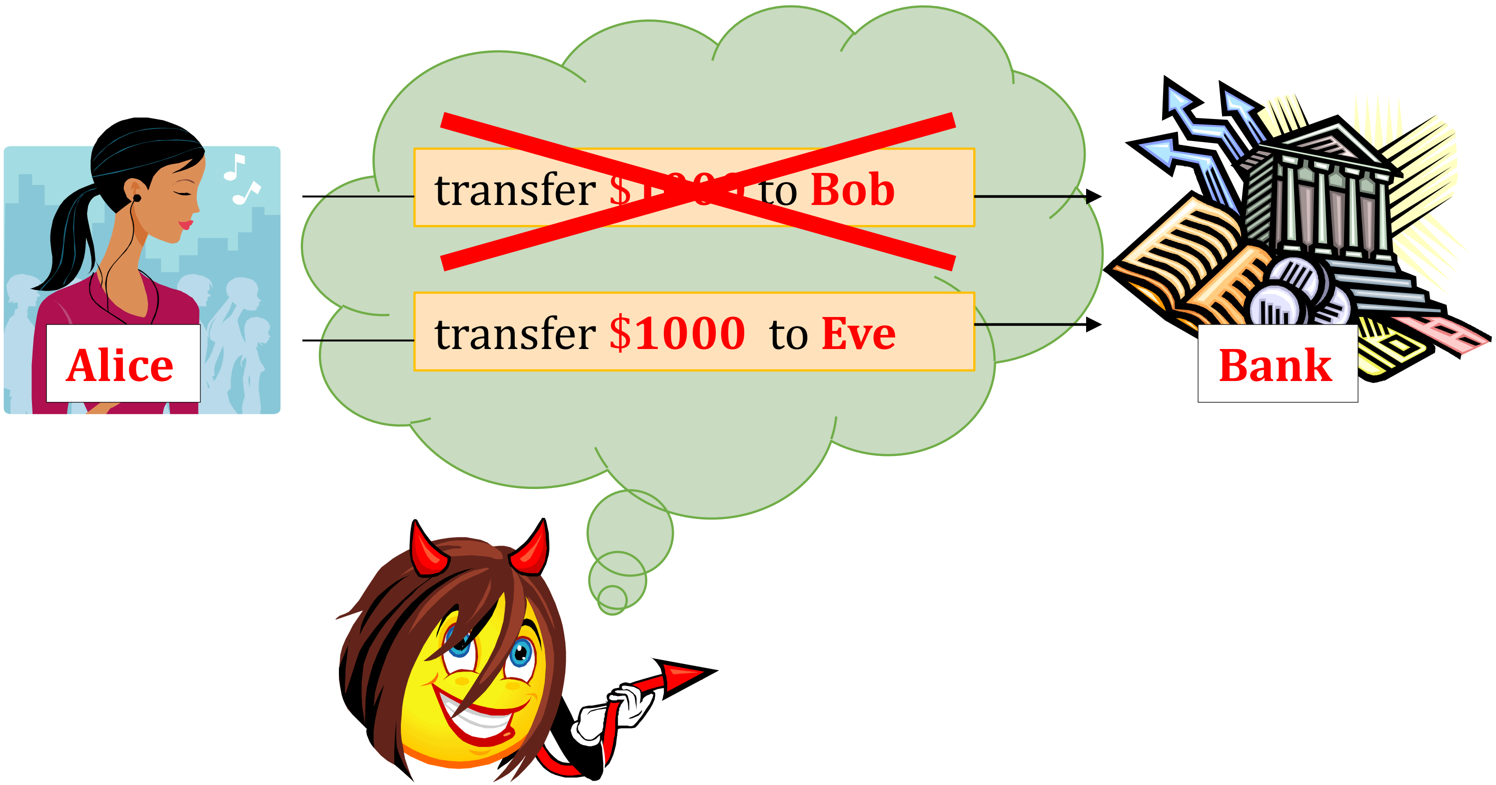
M



interferes with the transmission (modifies the message, or inserts a new one)

How can **Bob** be sure that ***M*** **really comes from Alice**?

Sometimes more important than secrecy!



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

Does encryption guarantee message integrity?

<u>Idea</u>:

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

<u>Hope</u>: only **Alice** knows $\boldsymbol{k}$, so nobody else can produce a valid ciphertext.

<u>This doesn't work!</u>

<u>Example</u>: one-time pad.

plaintext $\boldsymbol{m}$	transfer **$1000** to **Bob**
key $\boldsymbol{k}$	
xor	
ciphertext $\boldsymbol{c}$	

If **Eve** knows $\boldsymbol{m}$ and $\boldsymbol{c}$ then she can calculate $\boldsymbol{k}$ and produce a ciphertext of any other message

What do we need?

A separate tool for **authenticating messages**.

This tool will be called

Message Authentication Codes
(MACs)

A **MAC** is a pair of algorithms

(Tag, Vrfy)

"tagging" algorithm

"verification algorithm"

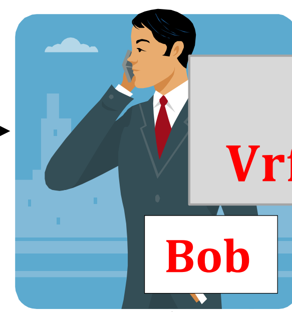
Message Authentication Codes

Eve can see $(\boldsymbol{m}, \boldsymbol{t} = \mathbf{Tag}_k(\boldsymbol{m}))$

She should not be able to compute a valid tag $\boldsymbol{t}'$ on any other message $\boldsymbol{m}'$.

$\boldsymbol{m}$

$(\boldsymbol{m}, \boldsymbol{t} = \mathbf{Tag}_k(\boldsymbol{m}))$



checks if

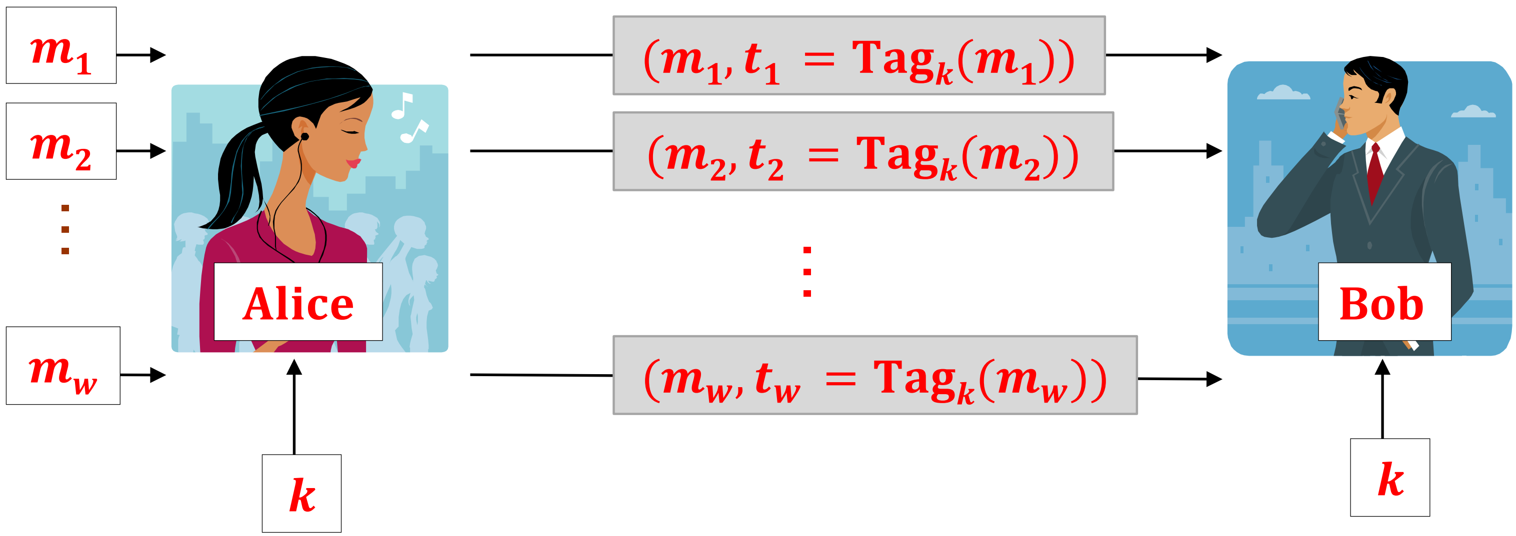
$\mathbf{Vrfy}_k(\boldsymbol{m}, \boldsymbol{t}) \in \{\mathbf{yes}, \mathbf{no}\}$

$\boldsymbol{k}$

$\boldsymbol{k}$

$\boldsymbol{k}$ is chosen randomly from some set $\mathcal{K}$

Message authentication – multiple messages



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

A mathematical view

$\mathcal{K}$ – **key** space

$\mathcal{M}$ – **plaintext** space

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

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

- **Tag:** $\mathcal{K} \times \mathcal{M} \rightarrow \mathcal{T}$ is a **tagging** algorithm,
- **Vrfy:** $\mathcal{K} \times \mathcal{M} \times \mathcal{T} \rightarrow$ **{yes, no}** is a **verification** algorithm.

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

Correctness

it always holds that:

$$\mathbf{Vrfy}_k(\boldsymbol{m}, \mathbf{Tag}_k(\boldsymbol{m})) = \textbf{yes}.$$

Conventions

If $\mathbf{Vrfy}_k(\boldsymbol{m}, \boldsymbol{t}) = \mathbf{yes}$ then we say that $\boldsymbol{t}$ is a **valid tag on the message** $\boldsymbol{m}$.

If **Tag** is **deterministic**, then **Vrfy** just computes **Tag** and compares the result.

In this case we do not need to define **Vrfy** explicitly.

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!

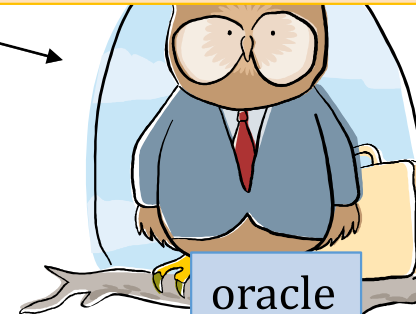
We assume that:

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 $m' \notin \{m_1, \dots, m_w\}$.

security parameter $\mathbf{1^n}$

adversary

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



oracle

$\boldsymbol{m_1}$

$(\boldsymbol{m_1}, \boldsymbol{t} = \mathbf{Tag}_k(\boldsymbol{m_1}))$

$\vdots$

$\boldsymbol{m_w}$

$(\boldsymbol{m_w}, \boldsymbol{t} = \mathbf{Tag}_k(\boldsymbol{m_w}))$

We say that the adversary **breaks the MAC scheme** at the end **she outputs** $(\boldsymbol{m'}, \boldsymbol{t'})$ such that

$$\mathbf{Vrfy}_k(\boldsymbol{m'}, \boldsymbol{t'}) = \mathbf{yes}$$

and

$$\boldsymbol{m'} \notin \{\boldsymbol{m_1}, \ldots, \boldsymbol{m_w}\}$$

The security definition

We say that **(Tag, Vrfy)** is **secure** if

$\forall$

polynomial-time adversary **A**

P(*A* breaks it) is negligible (in **n**)

Aren’t we too paranoid?

Maybe it would be enough to require that:

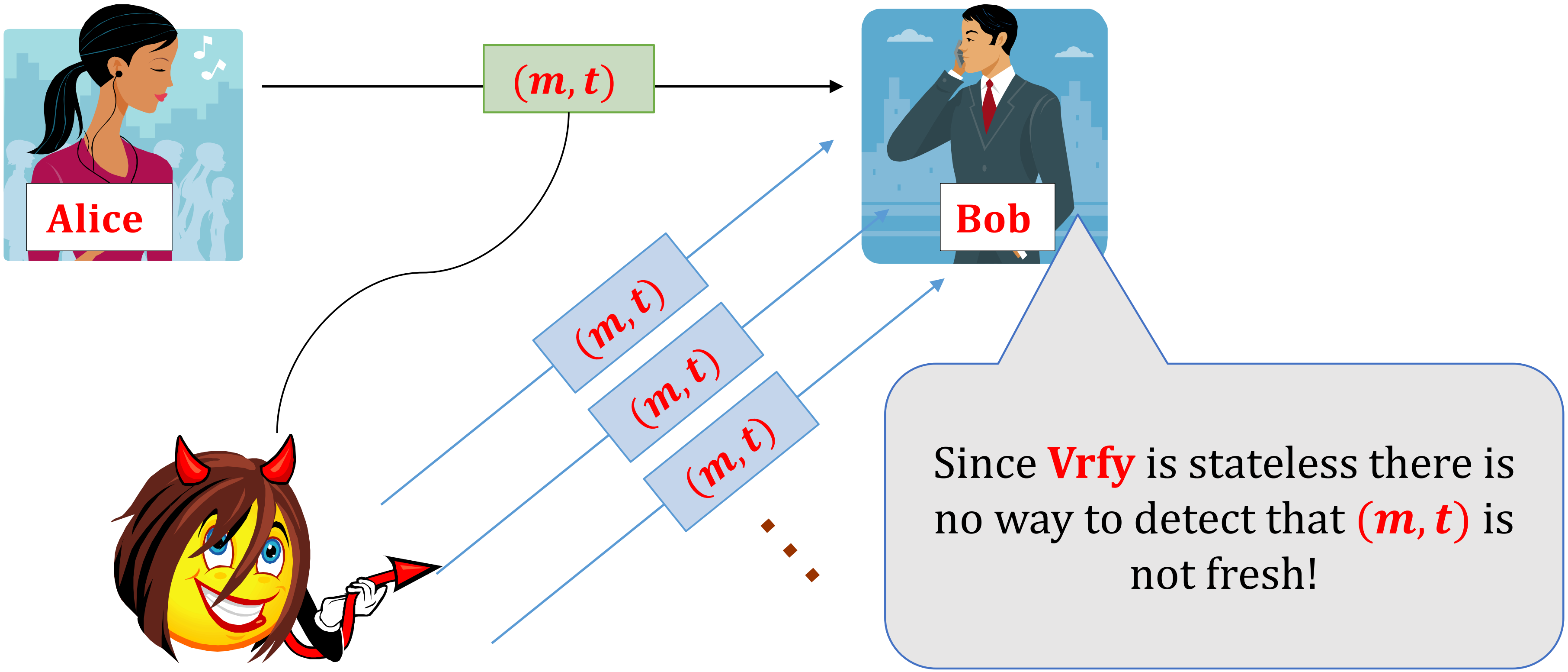
the adversary succeds only if he forges a message that “*makes sense*”.

(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**, **nonces**...).

Constructing a MAC

1. There exist **MACs** that are secure even if the adversary is **infinitely-powerful**.
   These constructions are **not practical**.

2. **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**).

Plan

1. Introduction to Message Authentication Codes (MACs).
2. Constructions of MACs from block ciphers

A simple construction from a block cipher

Let

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

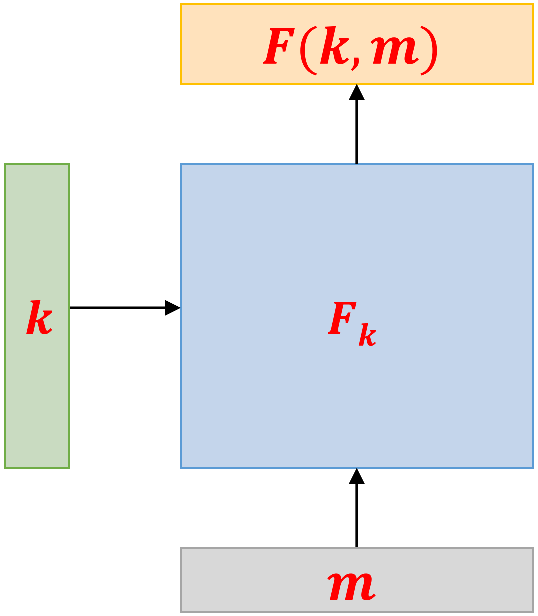
be a **block cipher** (a **PRF**).

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

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

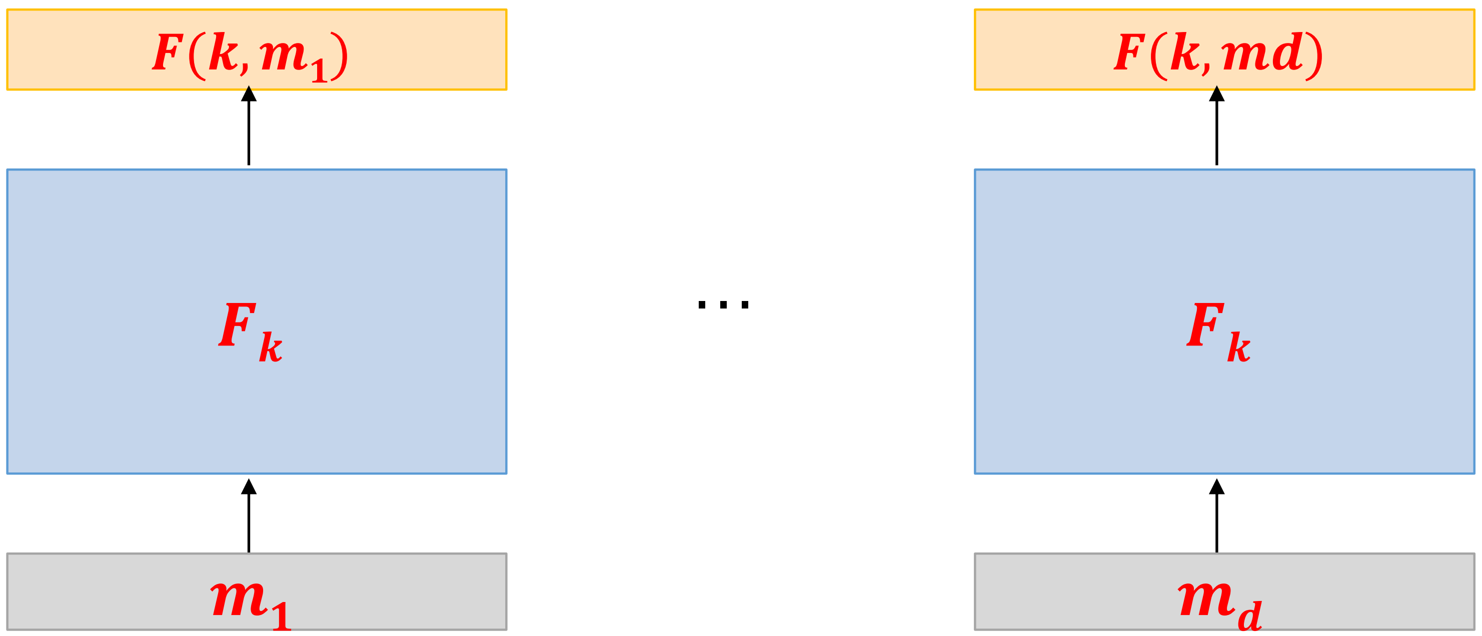
It can be proven that it is a secure **MAC**.

How to generalize it to longer messages?



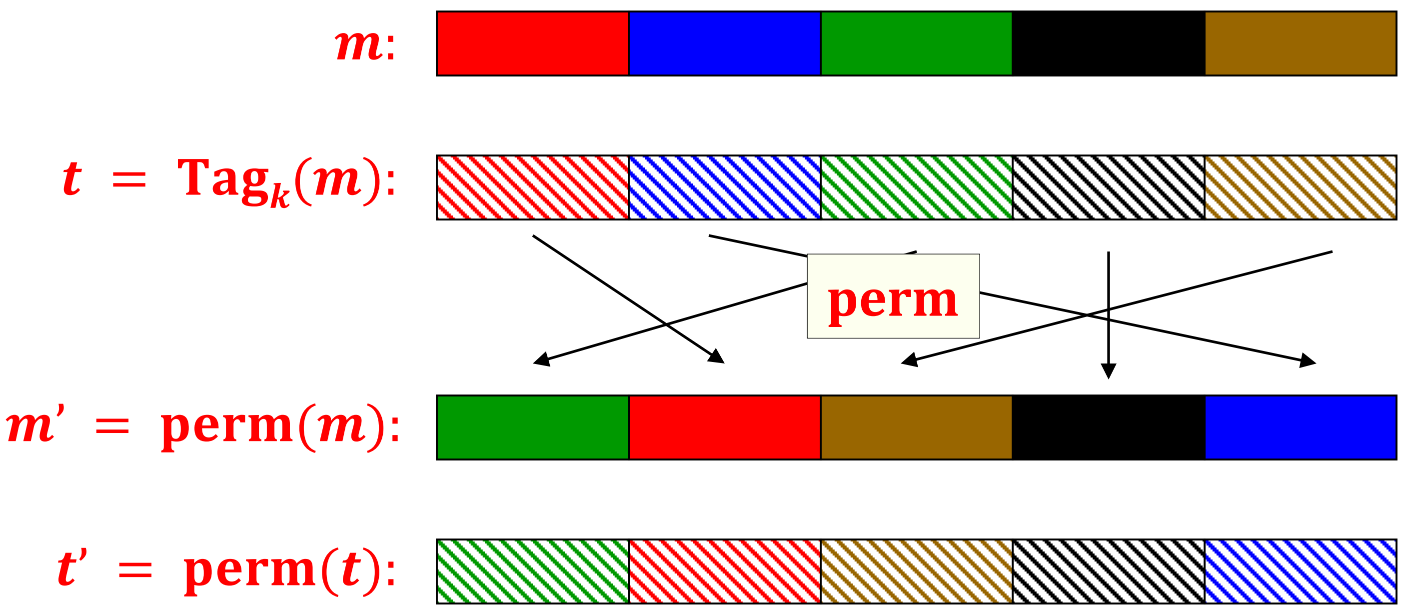
Idea 1

- divide the message in blocks $m_1, \dots, md$
- and authenticate each block separately



This doesn't work!

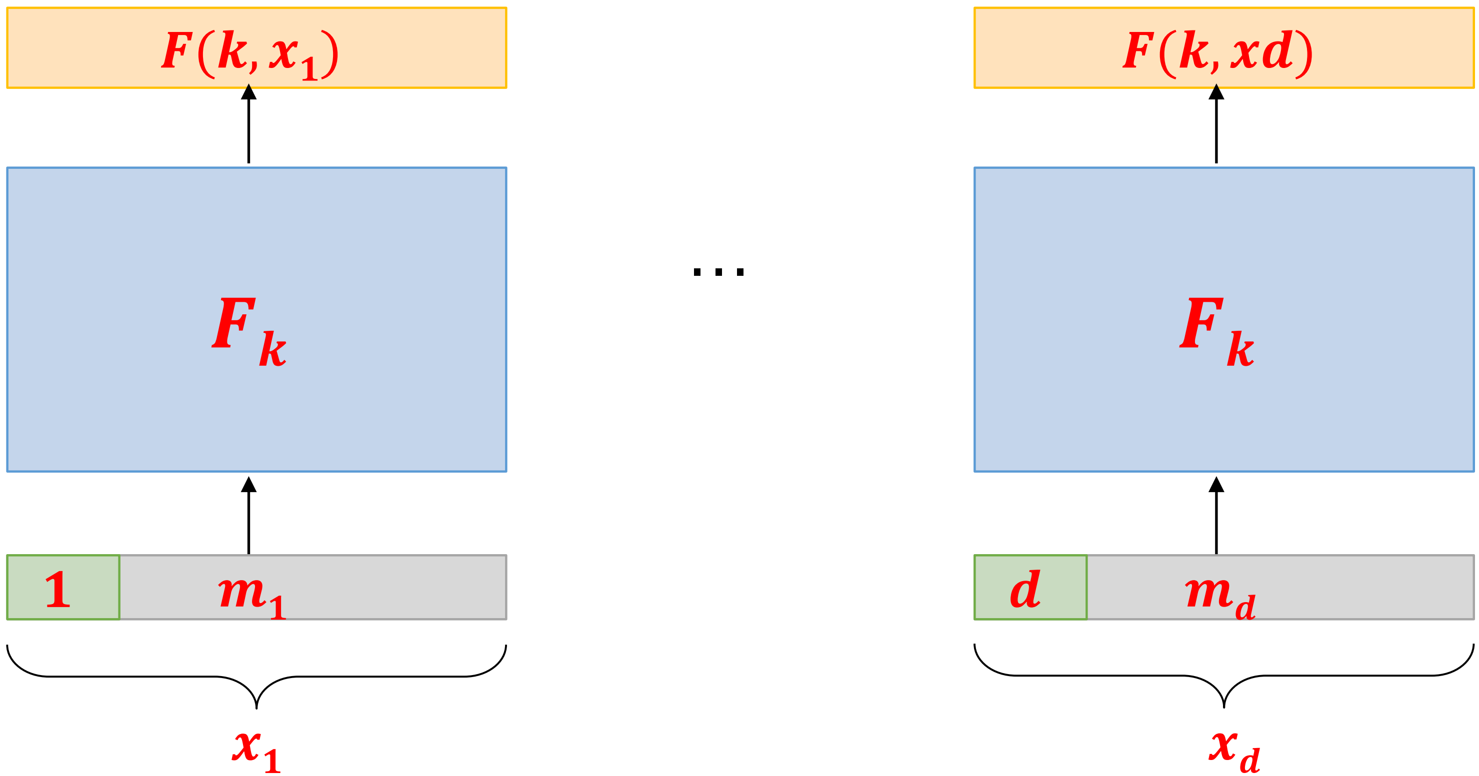
What goes wrong?



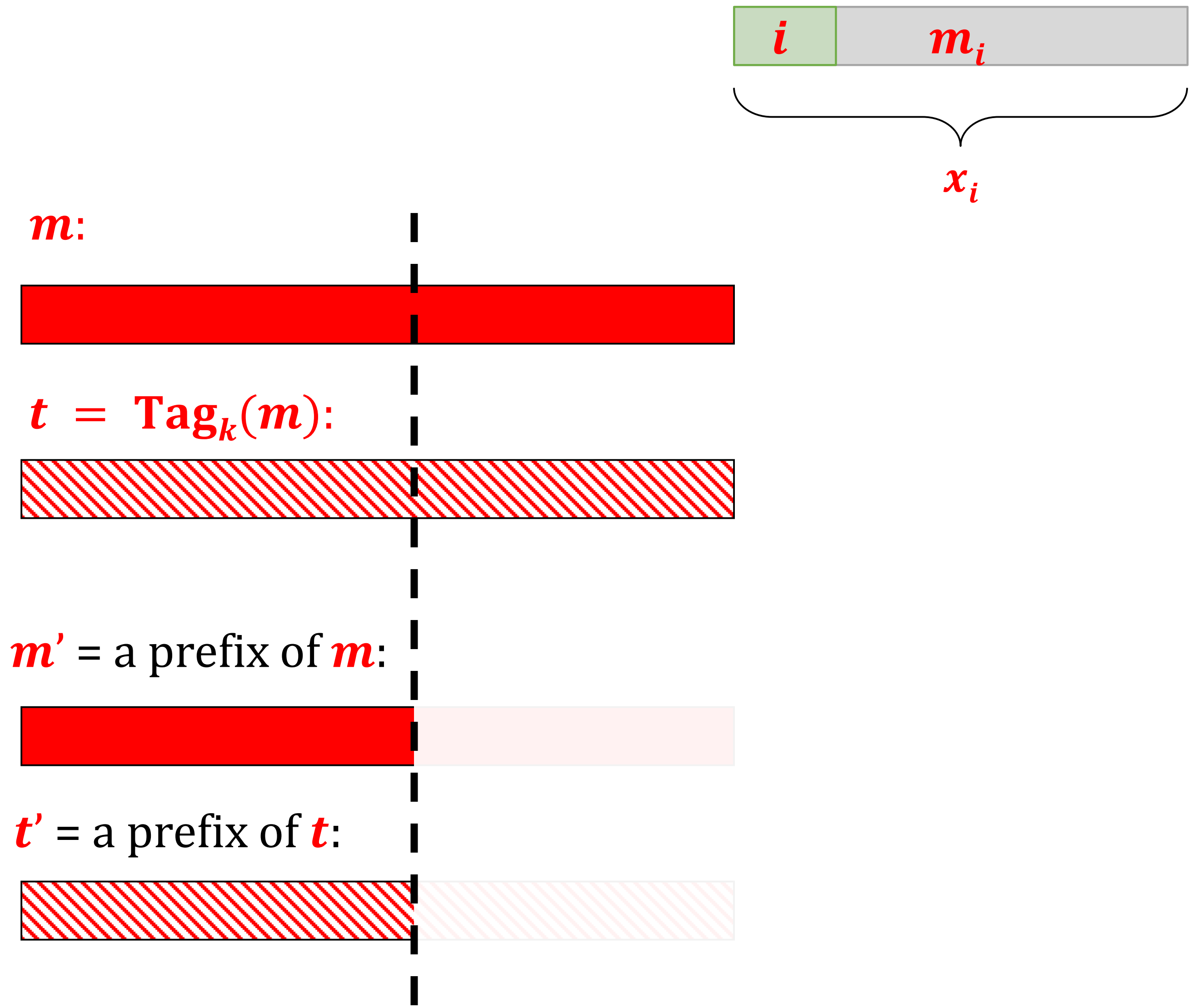
Then **t’** is a valid tag on m'.

Idea 2

Add a counter to each block.



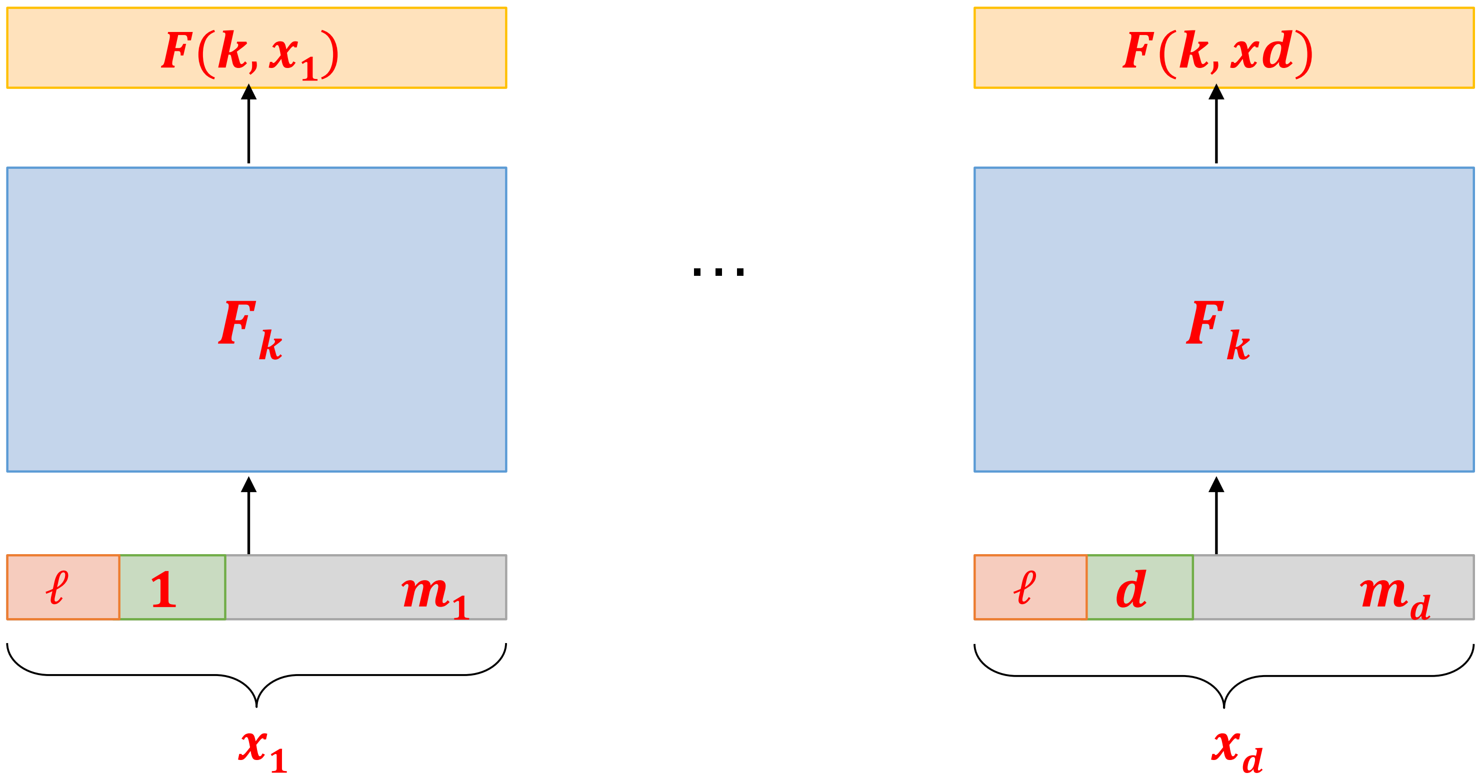
This doesn't work either!



Then t' is a valid tag on m'.

Idea 3

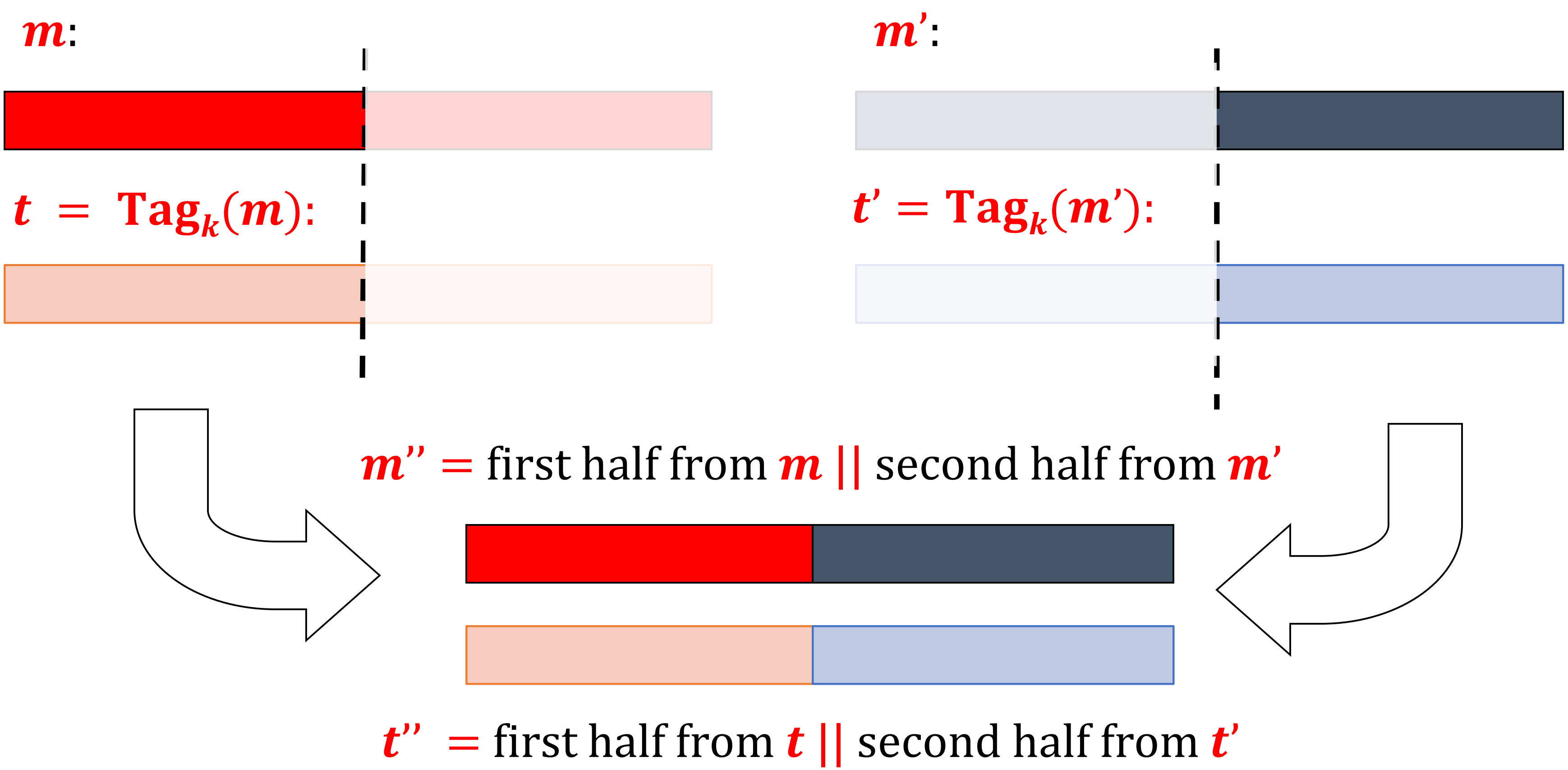
Add $\ell := |m|$ to each block



This doesn't work either!

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

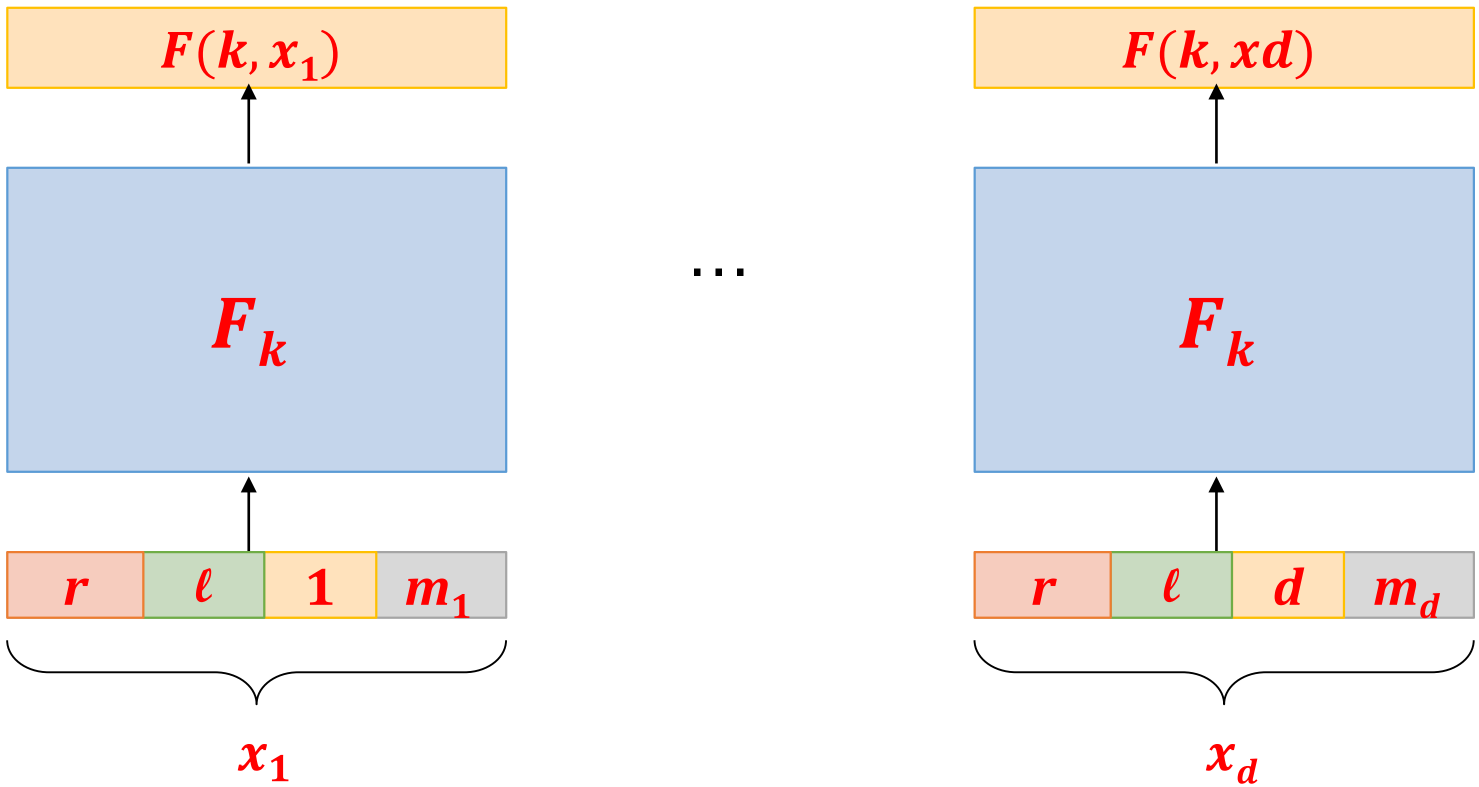
What goes wrong?



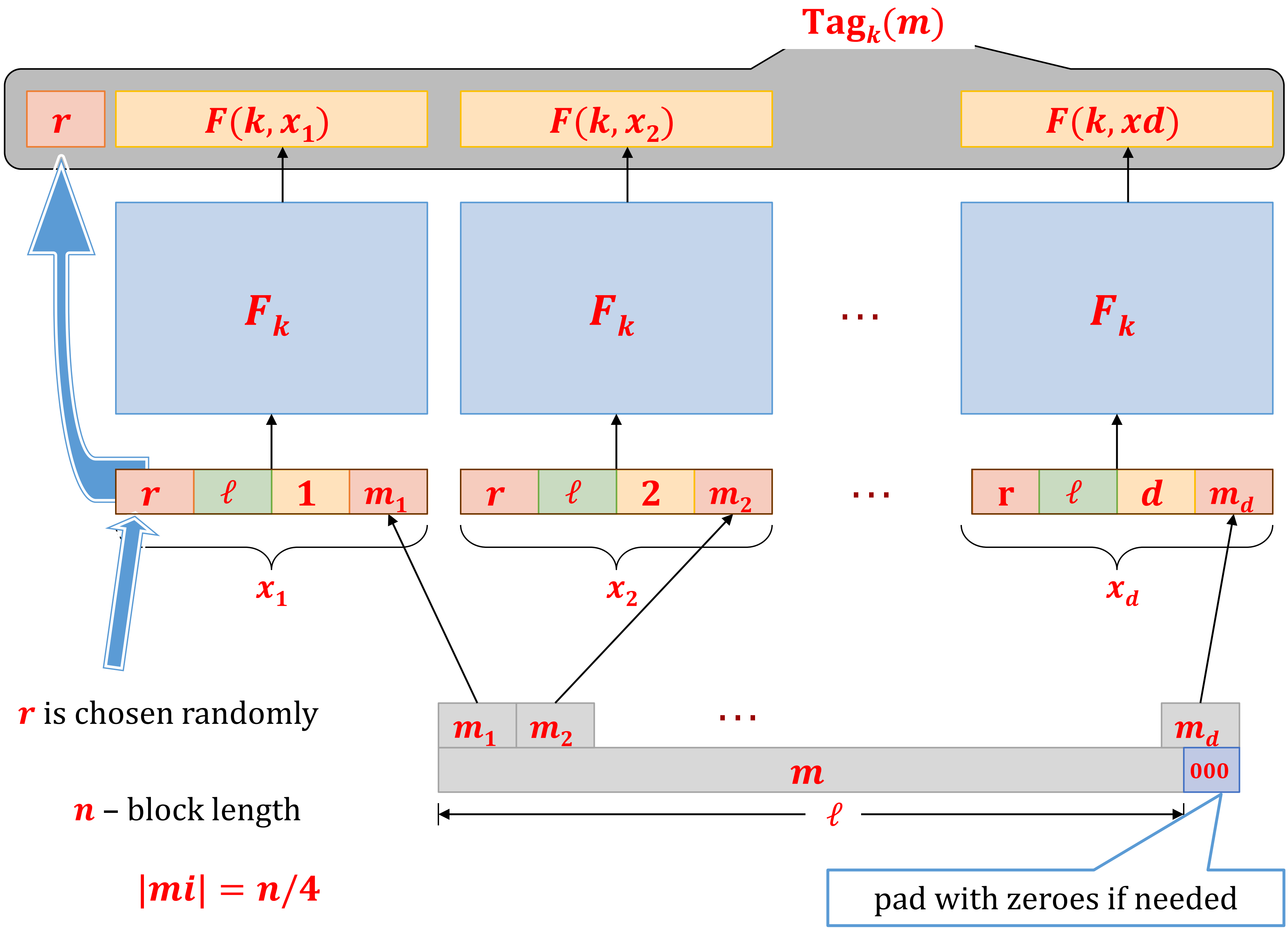
Then t'' is a valid tag on m''.

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, xd)
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
n – block length
|mi| = n/4
m_1 m_2 ... m_d
m
000
ℓ
pad with zeroes if needed

This construction can be proven secure

Theorem

Assuming that

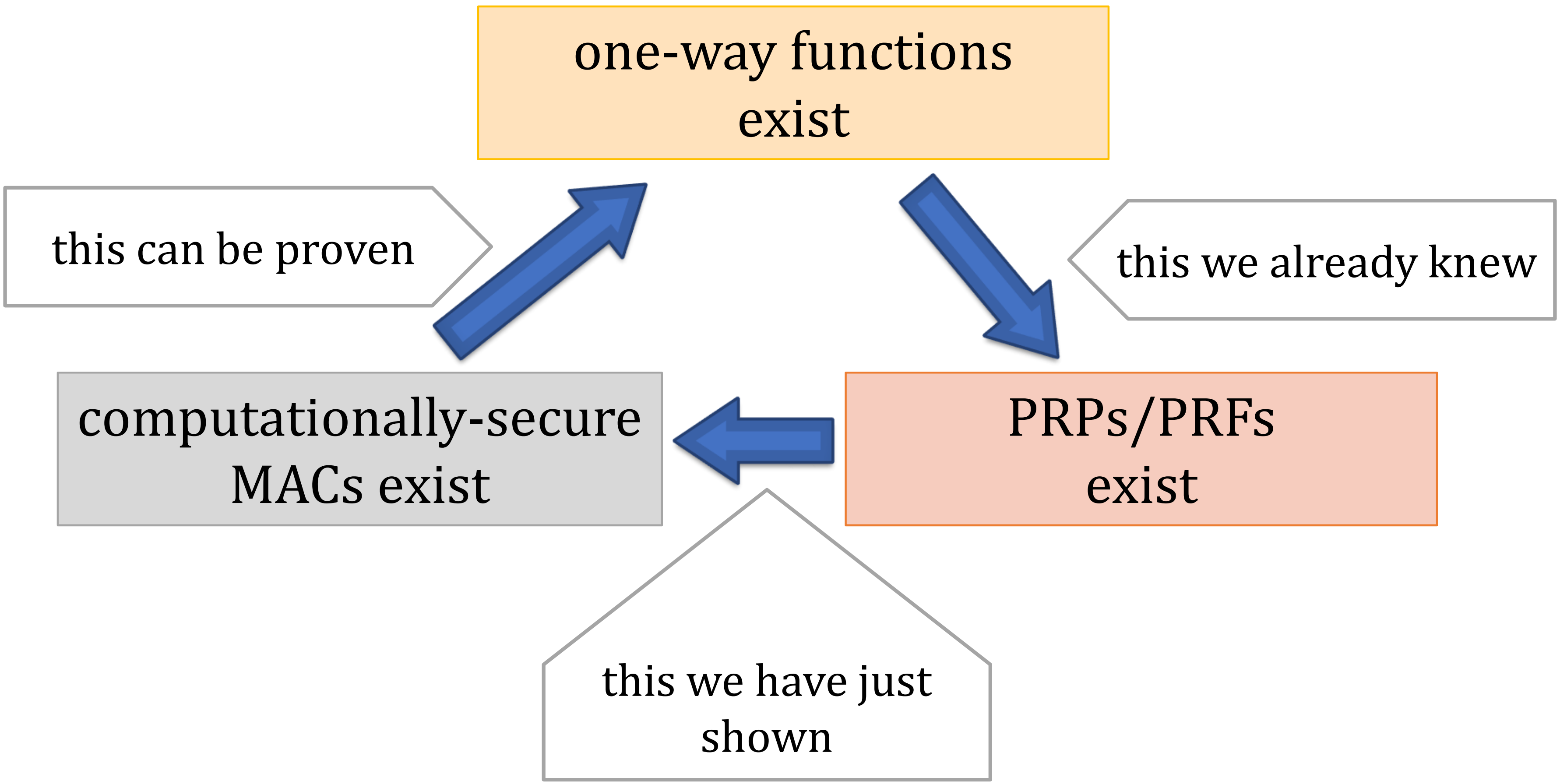
$\boldsymbol{F} : \{\mathbf{0},\mathbf{1}\}^{\boldsymbol{n}} \times \{\mathbf{0},\mathbf{1}\}^{\boldsymbol{n}} \rightarrow \{\mathbf{0},\mathbf{1}\}^{\boldsymbol{n}}$ is a **pseudorandom permutation**

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

Proof idea:

- Suppose it is **not** a secure **MAC**.
- Let $\boldsymbol{A}$ be an adversary that breaks it with a non-negligible probability.
- We construct a distinguisher $\boldsymbol{D}$ that distinguishes $\boldsymbol{F}$ from a random permutation.

A new member of “Minicrypt”



Our construction is not practical

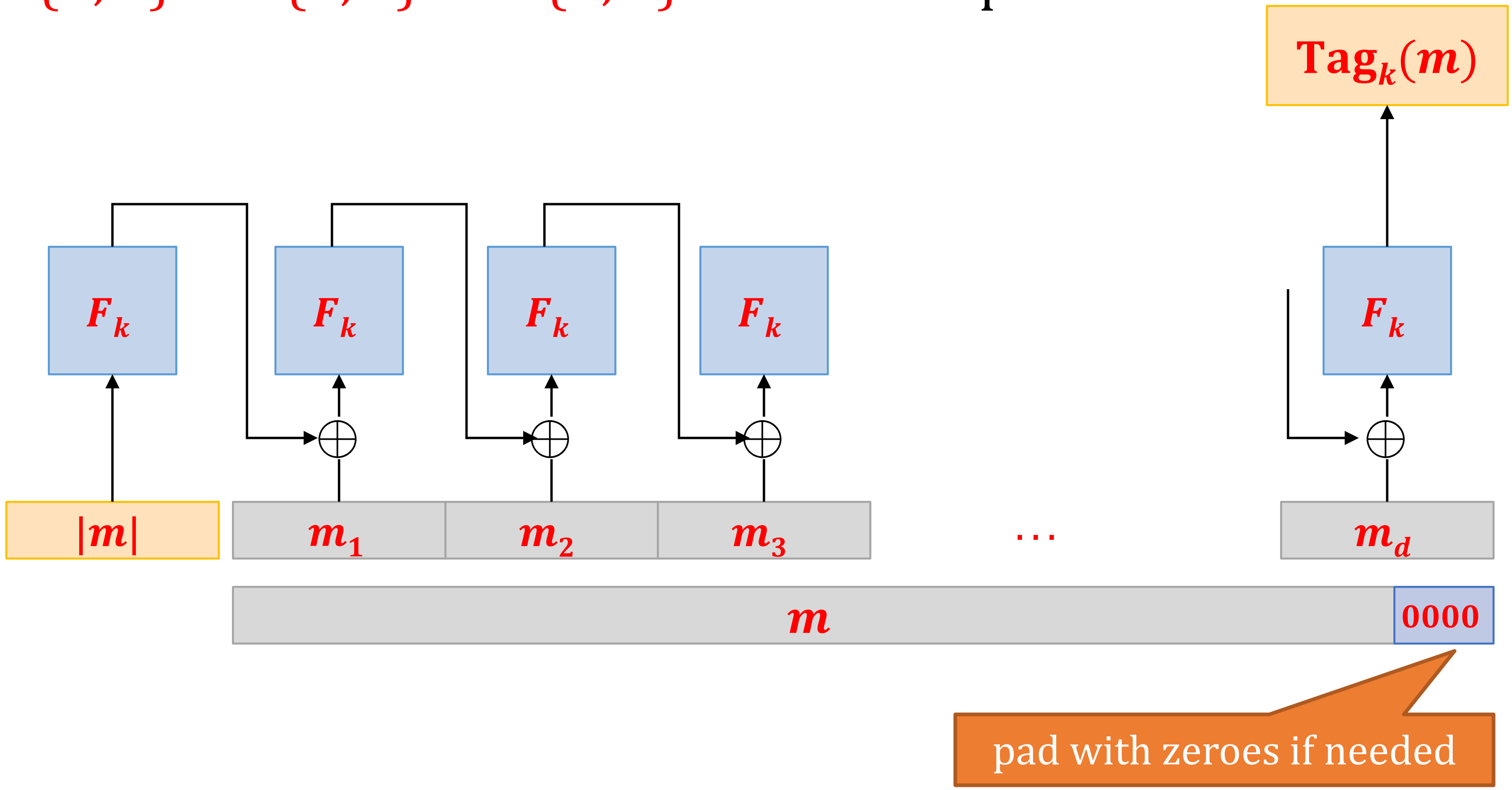
Problem:

The tag is **4 times longer** than the message...

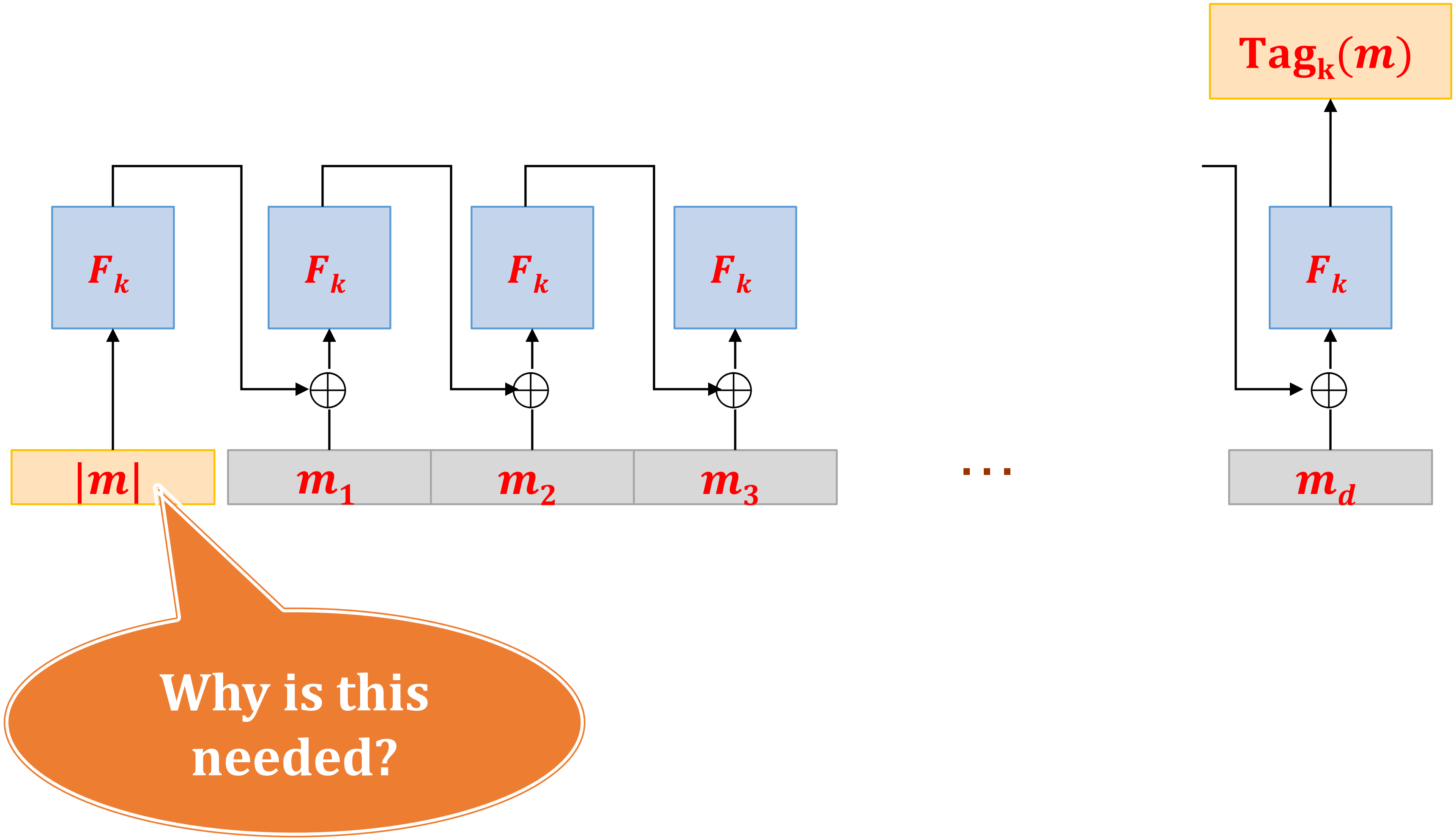
We can do much better!

CBC-MAC

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

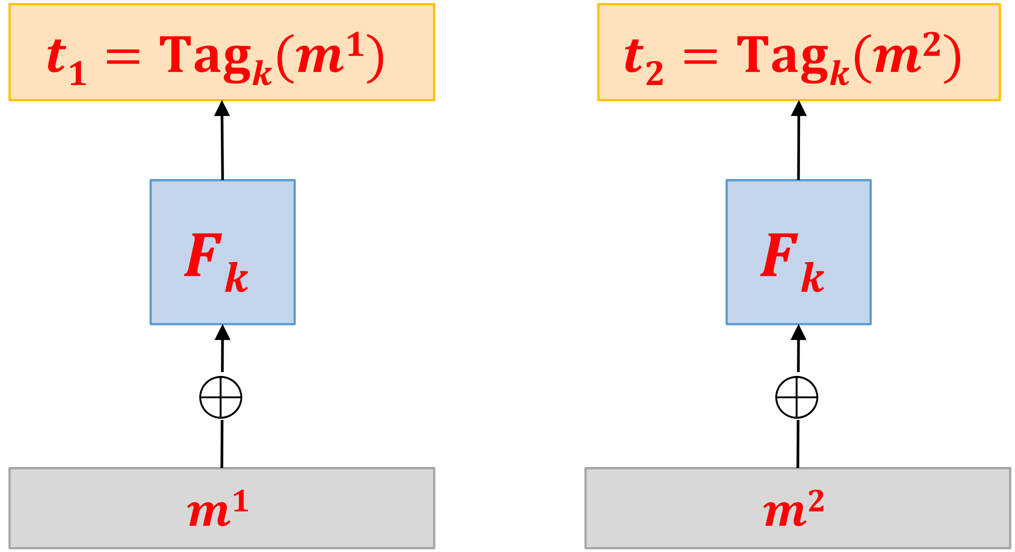


Other variants exist!



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

the adversary chooses:



now she can compute: