Lecture 5

Message Authentication and Server-Based Key Establishment

Stefan Dziembowski

www.crypto.edu.pl/Dziembowski

University of Warsaw

9.11.16

version 1.0

Secure communication

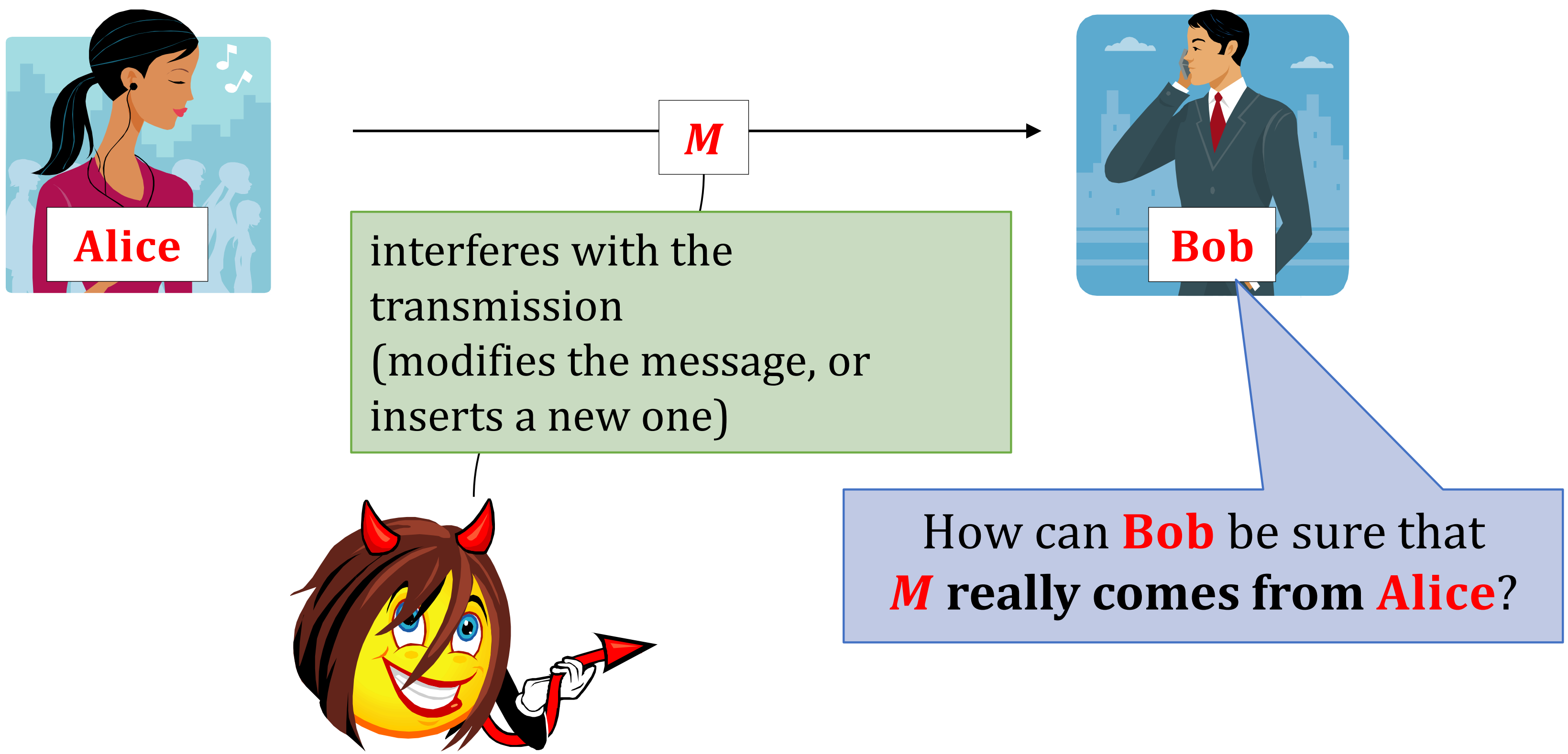
	encryption	authentication
private key	**1** private key encryption	**2** private key authentication
public key	**3** public key encryption	**4** signatures

Plan

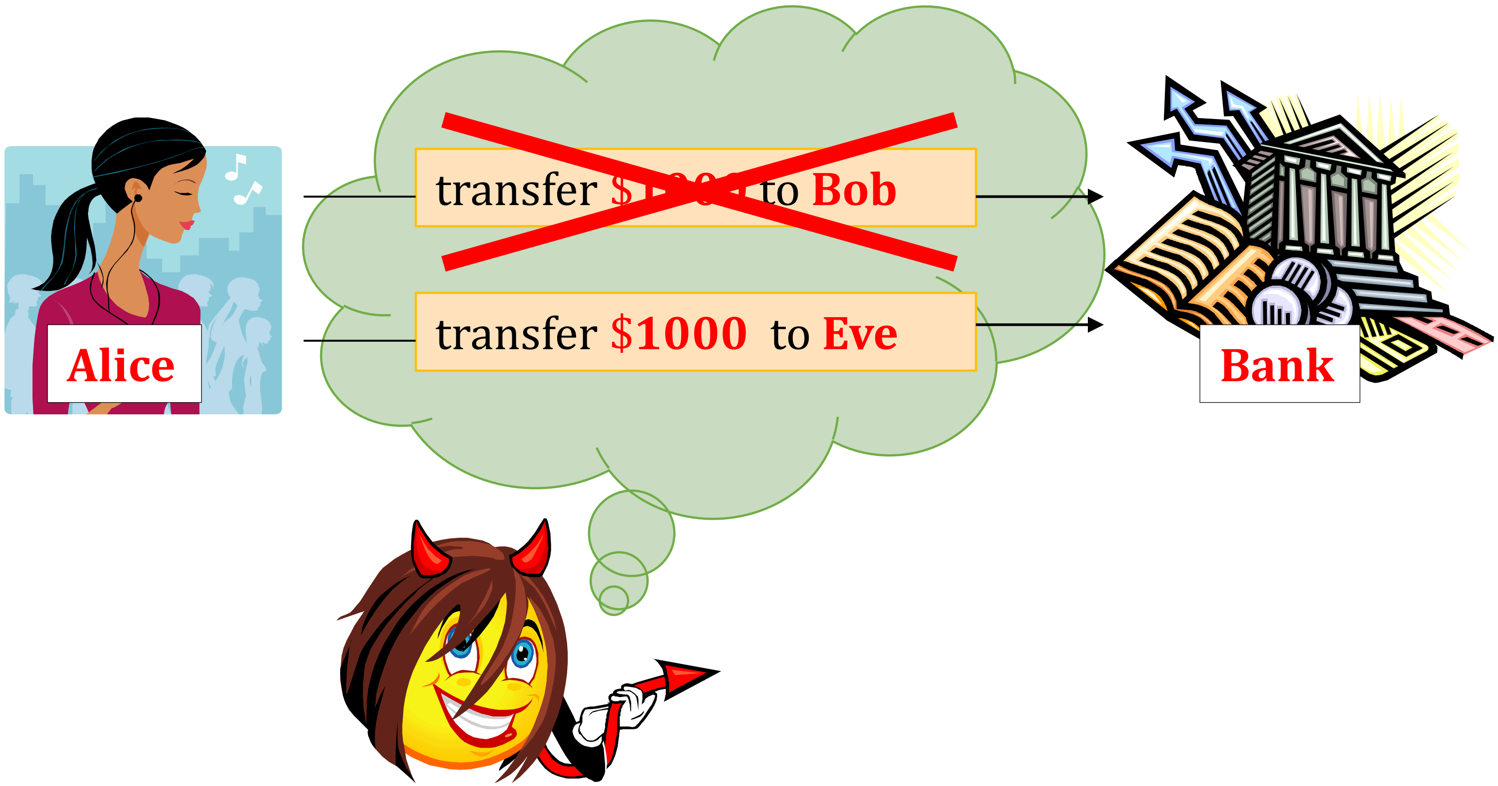
1. Introduction to Message Authentication Codes (MACs).
2. Constructions of MACs from block ciphers
3. Constructions of MACs from hash functions
4. Authenticated encryption
5. Key establishment with a trusted server
6. Outlook

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

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

This doesn't work!

Example: one-time pad.

plaintext m	transfer **$1000** to **Bob**
key k	
xor	
ciphertext c	

If **Eve** knows m and c then she can calculate 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

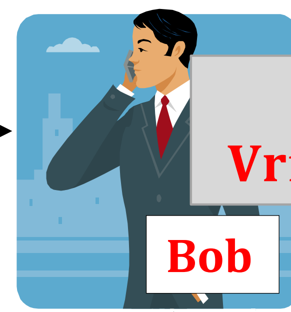
Eve can see $(m, t = \mathrm{Tag}_k(m))$

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

m

Alice

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



Bob

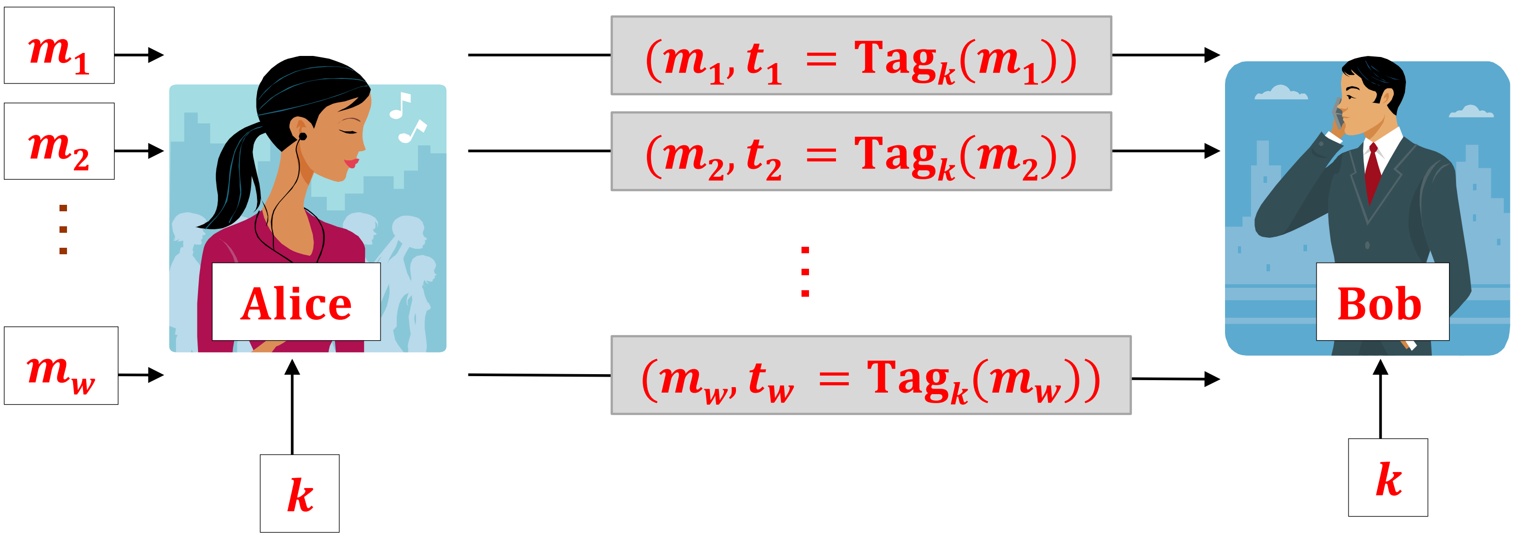
checks if
$\mathrm{Vrfy}_k(m, t) \in \{\mathrm{yes}, \mathrm{no}\}$

k

k

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 $\mathbf{Tag}_k(m)$ and $\mathbf{Vrfy}_k(m, t)$ instead of $\mathbf{Tag}(k, m)$ and $\mathbf{Vrfy}(k, m, t)$.

Correctness

it always holds that:

$\mathbf{Vrfy}_{\mathrm{k}}(m, \mathbf{Tag}_k(m)) = \mathbf{yes}$.

Conventions

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

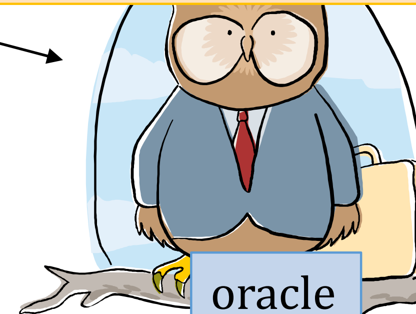
We assume that:

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

adversary

security parameter $\mathbf{1^n}$

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



oracle

$\boldsymbol{m_1}$

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

⋮

$\boldsymbol{m_w}$

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

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

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

and

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

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:

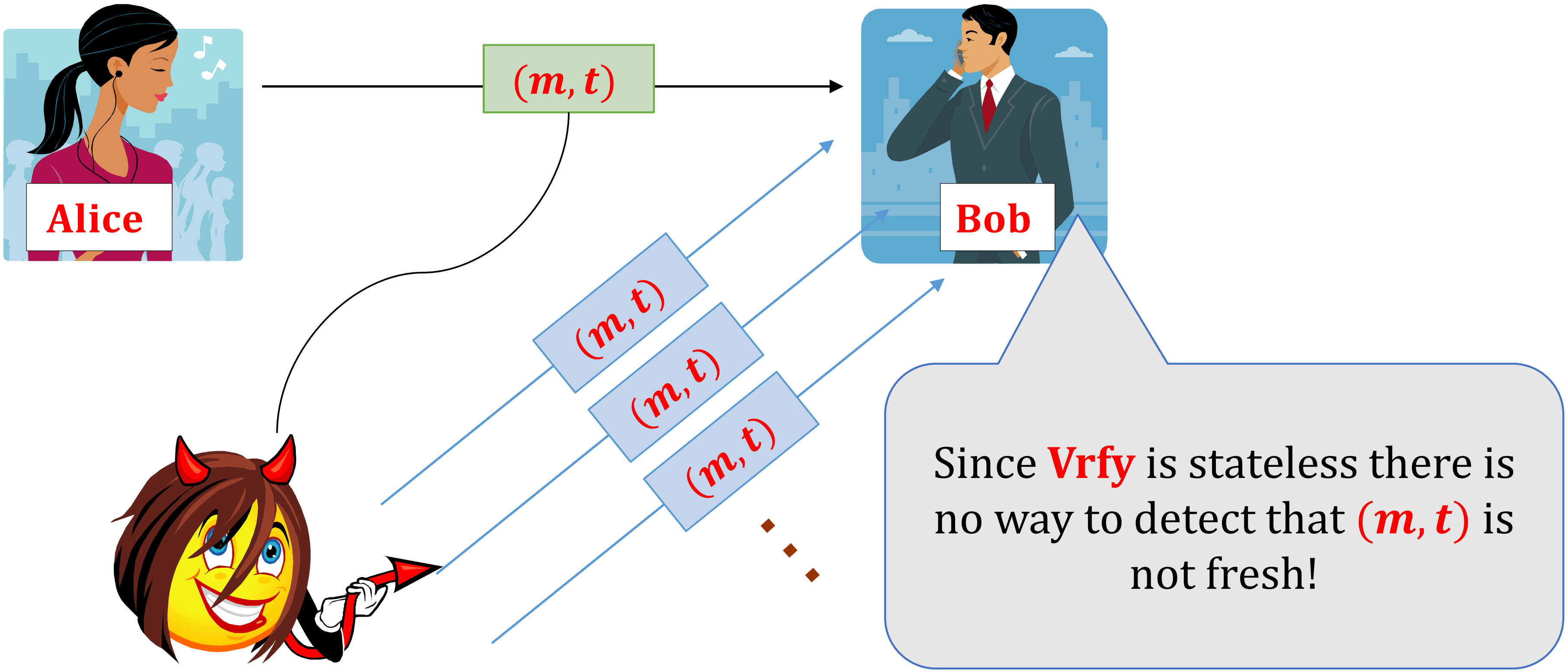
the adversary succeds only if he forges a message that "*makes sense*".

(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
3. Constructions of MACs from hash functions
4. Authenticated encryption
5. Key establishment with a trusted server
6. Outlook

A simple construction from a block cipher

Let

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

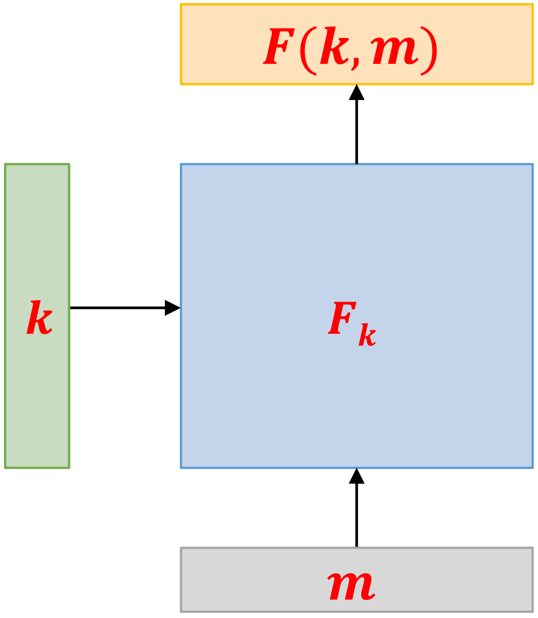
be a **block cipher** (a **PRF**).

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

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

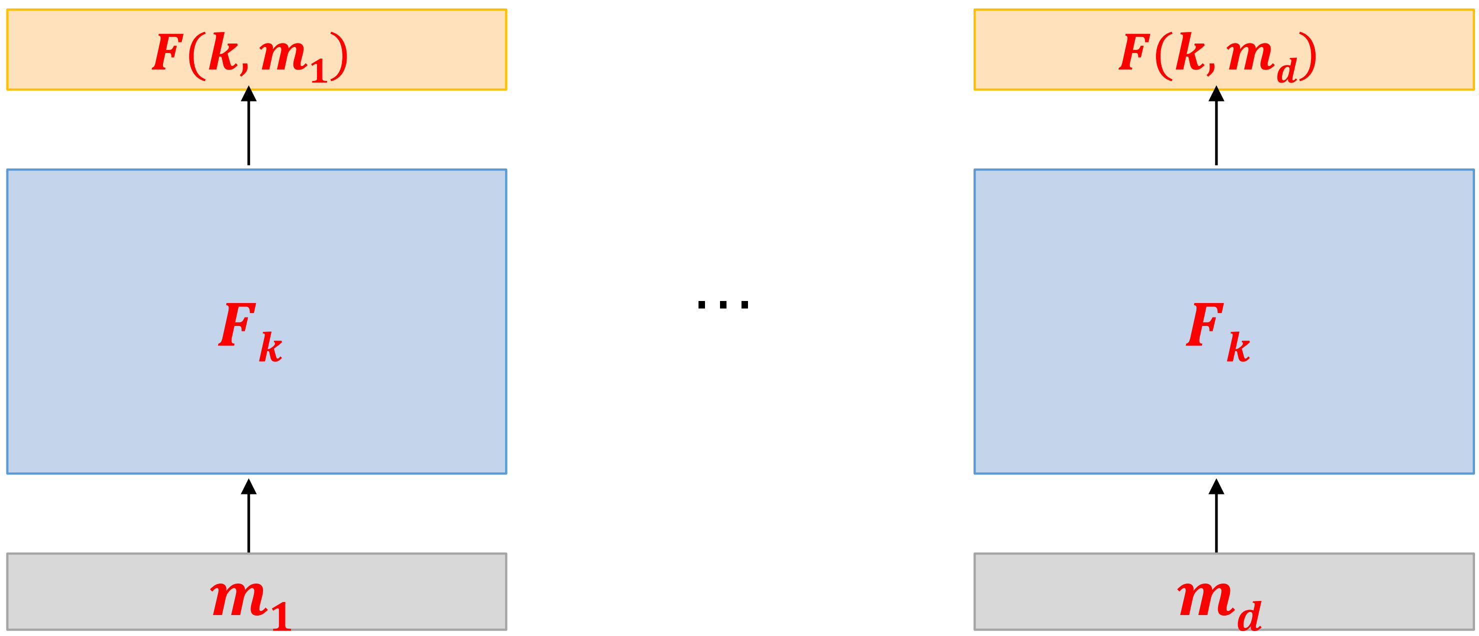
It can be proven that it is a secure **MAC**.

How to generalize it to longer messages?



Idea 1

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



This doesn't work!

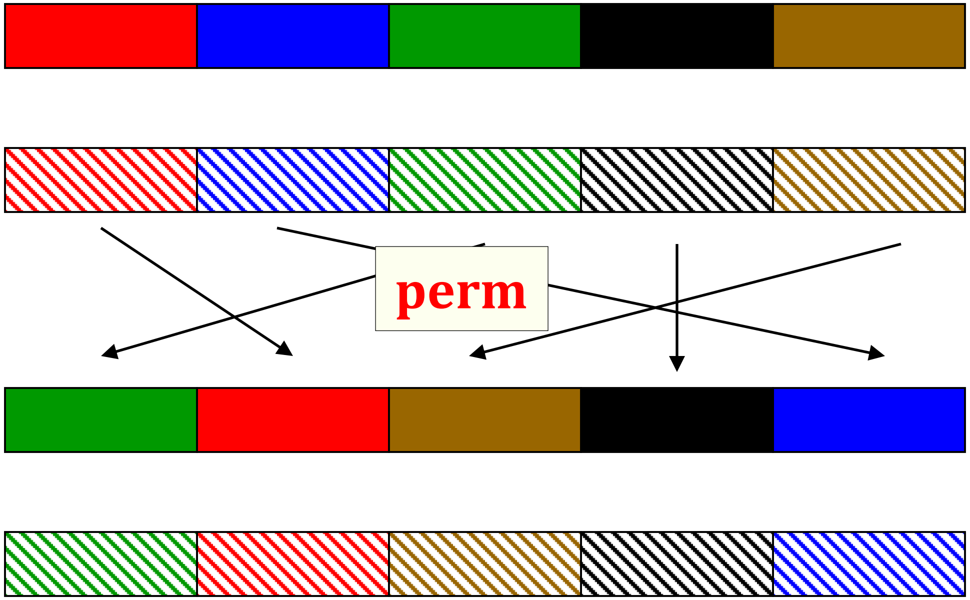
What goes wrong?

m:

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

$m' = \mathbf{perm}(m)$:

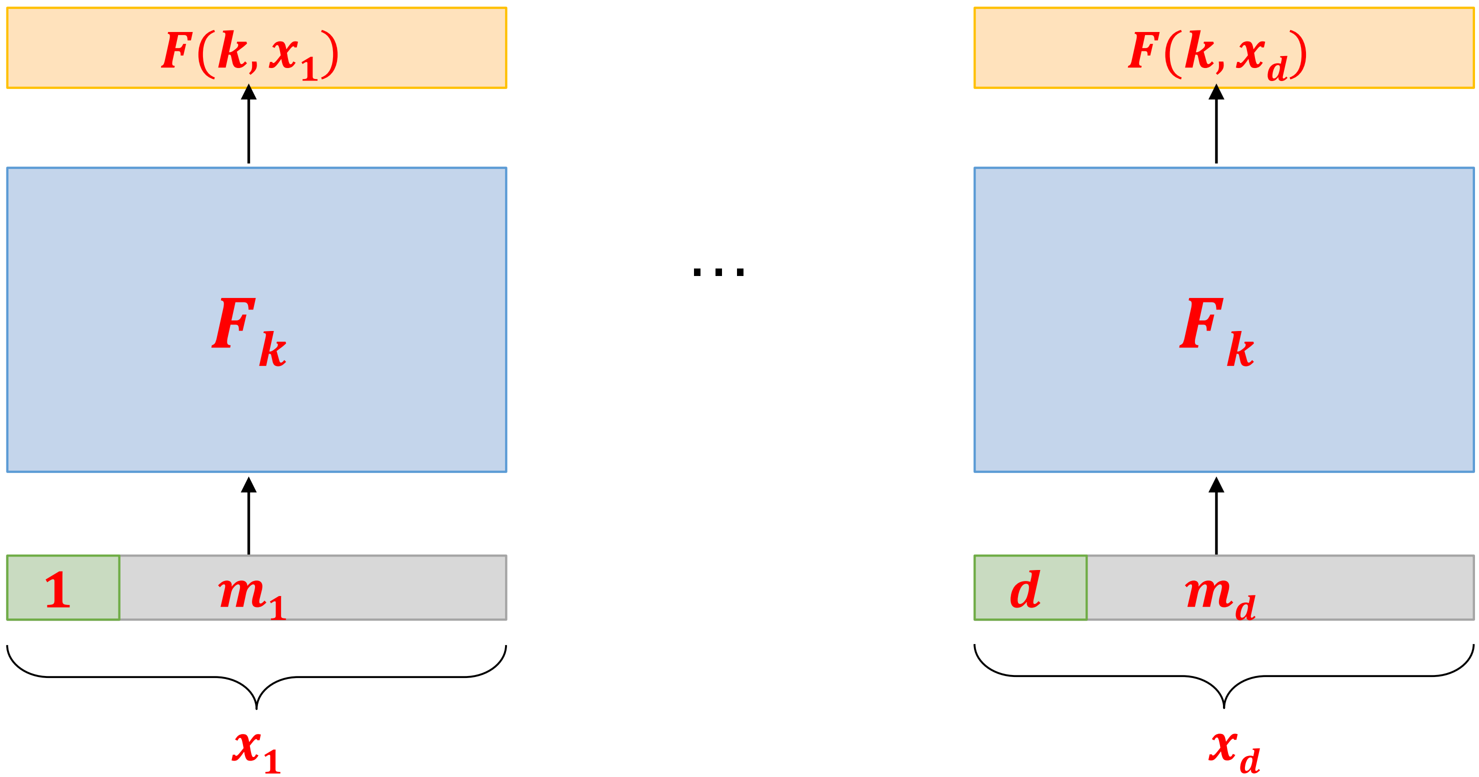
$t' = \mathbf{perm}(t)$:



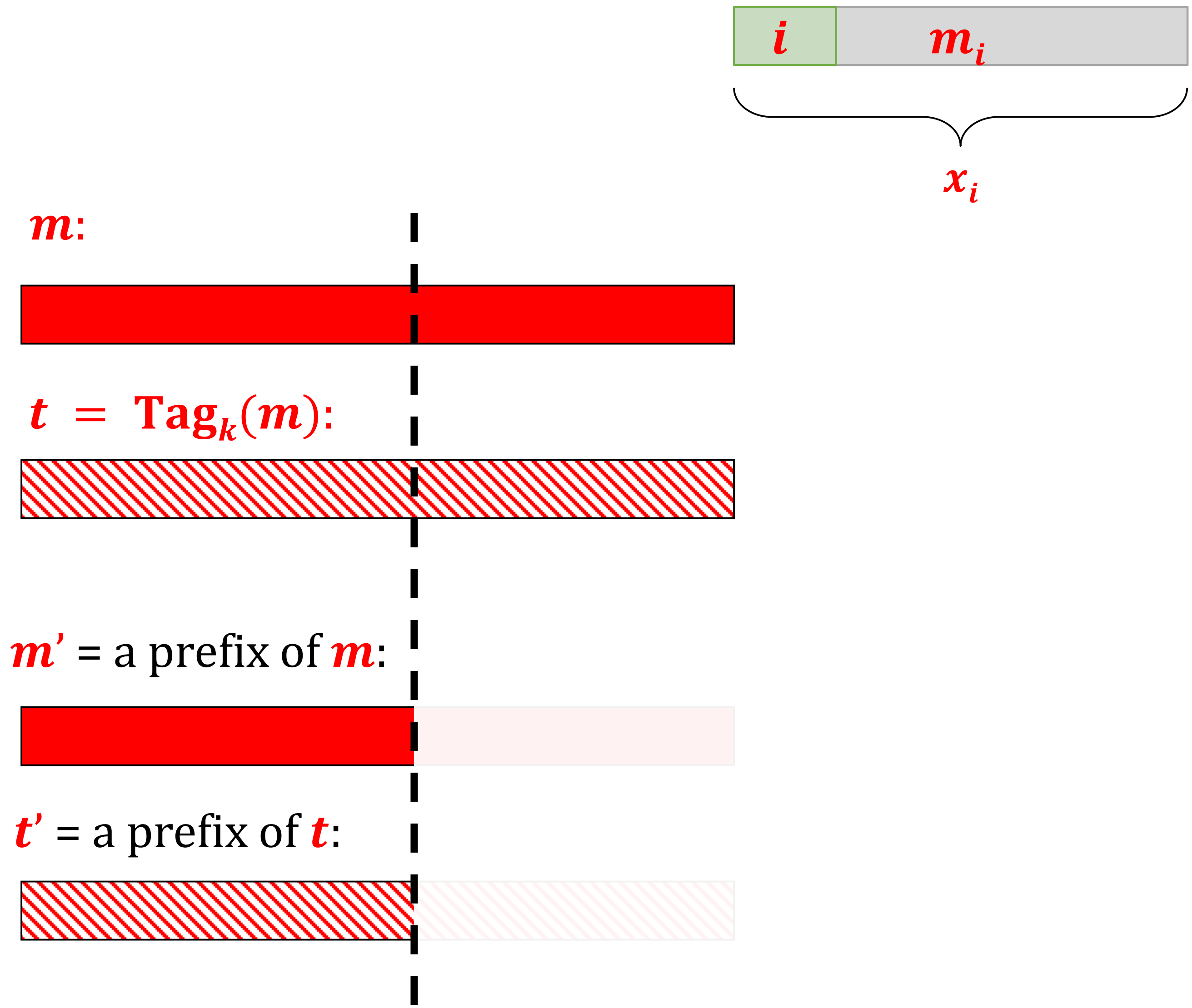
Then **t'** is a valid tag on m'.

Idea 2

Add a counter to each block.



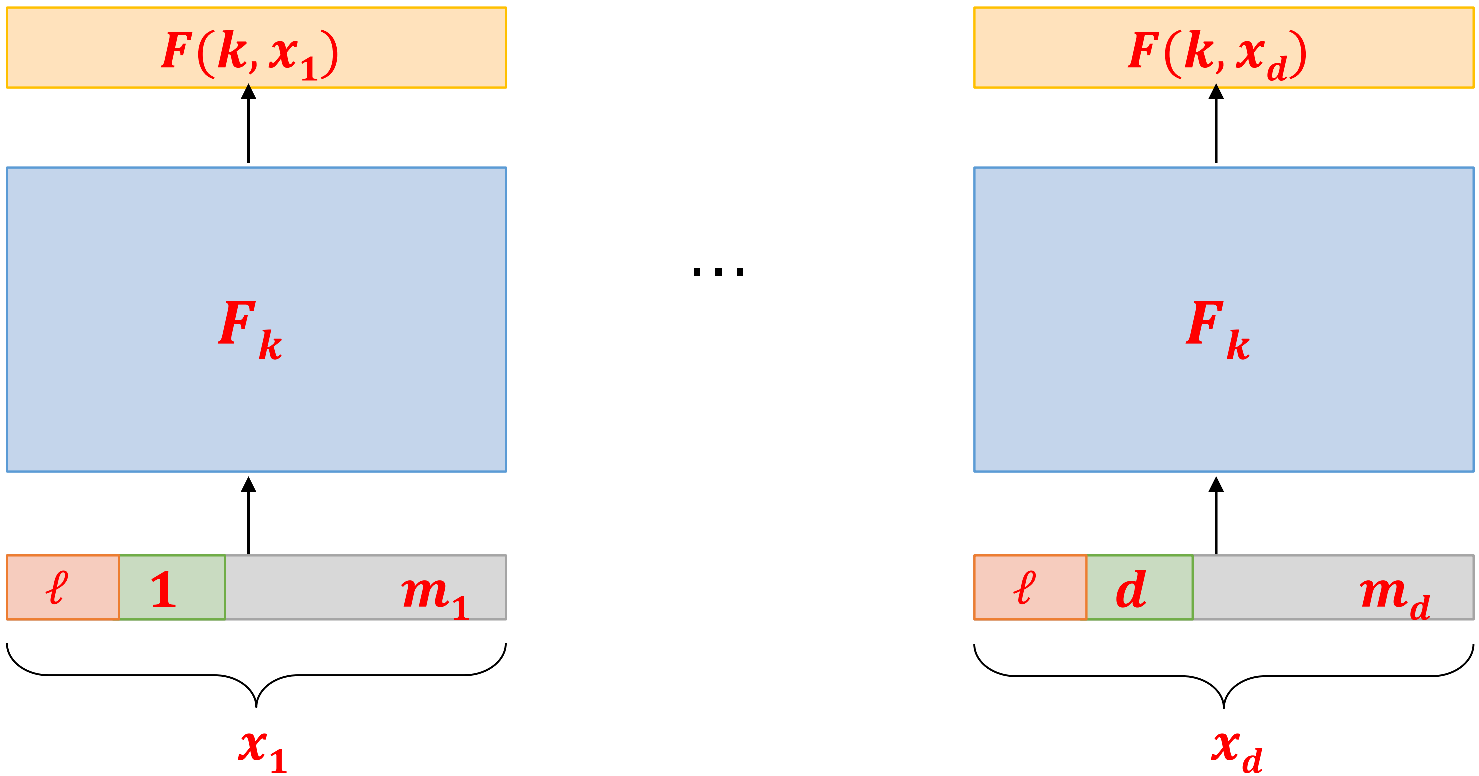
This doesn't work either!



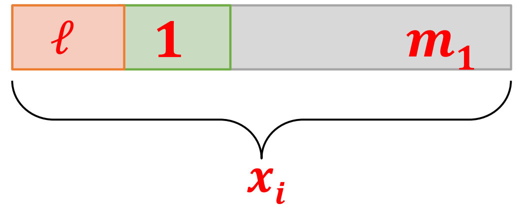
Then t' is a valid tag on m'.

Idea 3

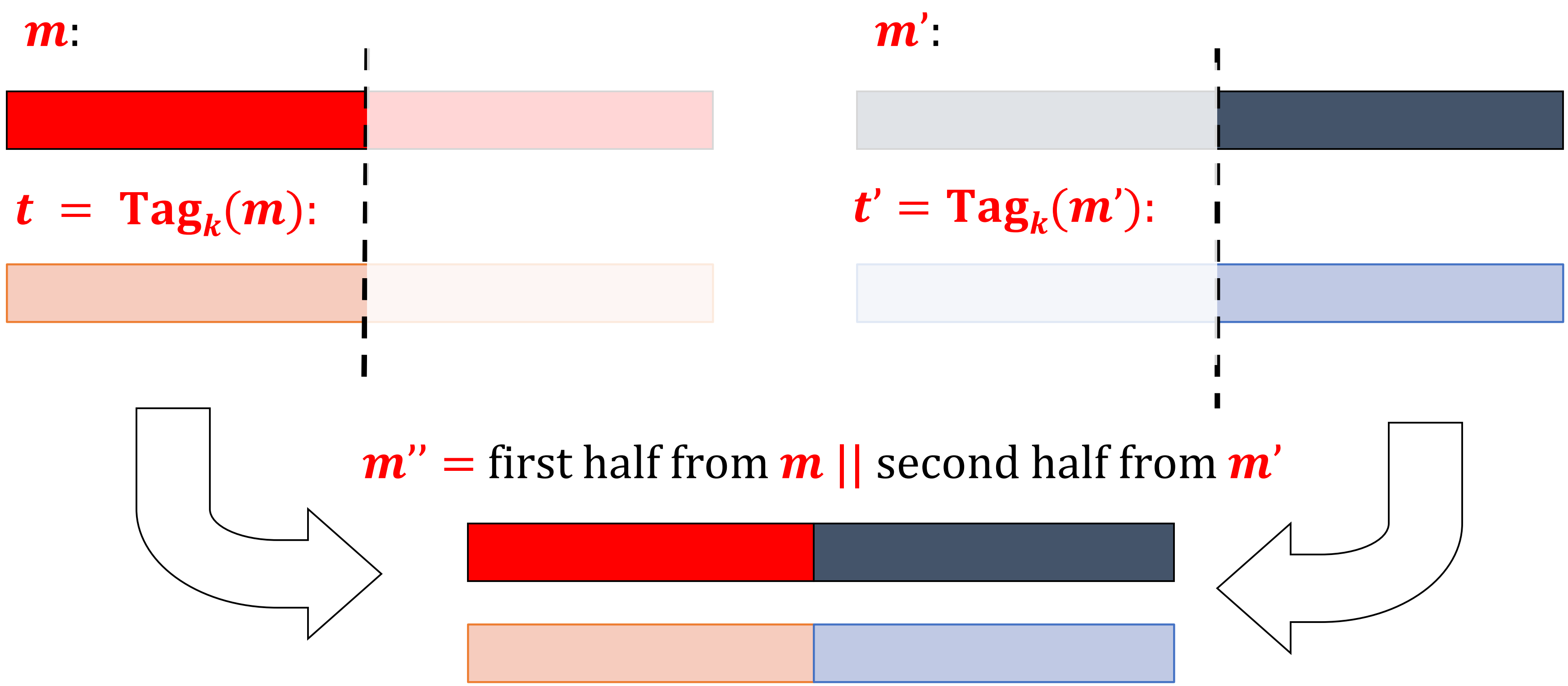
Add $\ell := |\boldsymbol{m}|$ to each block



This doesn't work either!



What goes wrong?

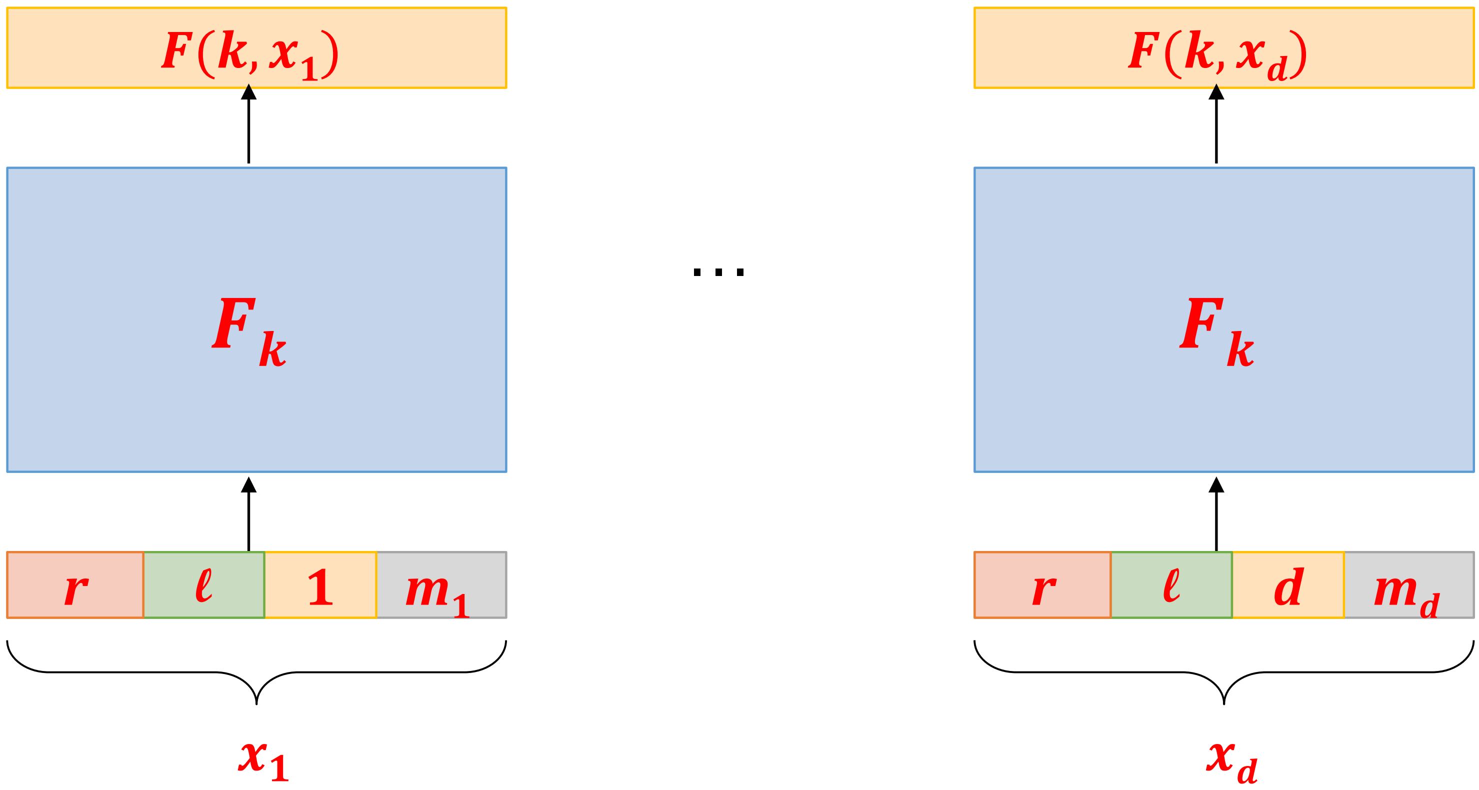


t'' = first half from t || second half from t'

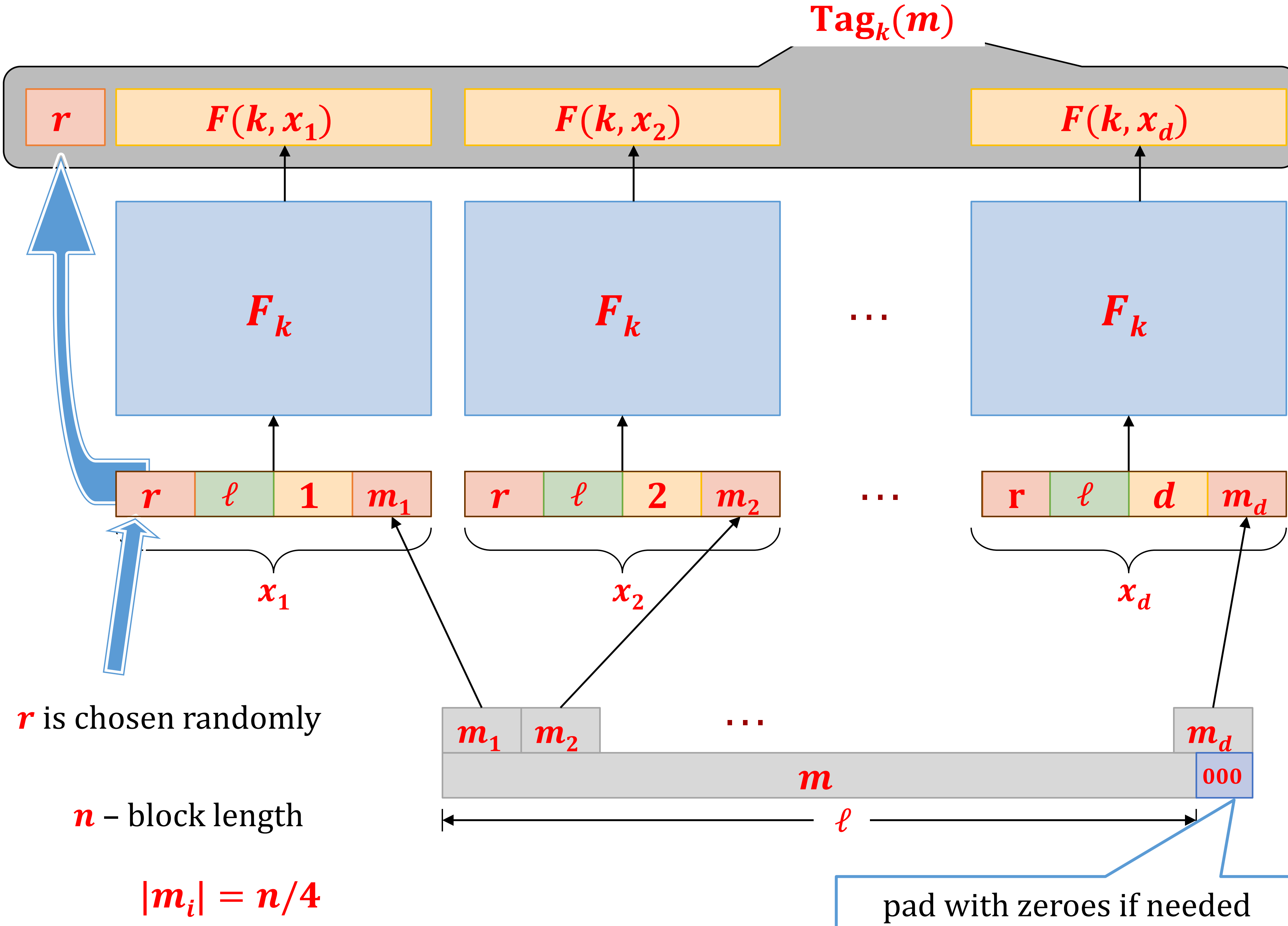
Then t'' is a valid tag on m''.

Idea 4

Add a fresh random value to each block!



This works!



r is chosen randomly

n – block length

$|m_i| = n/4$

This construction can be proven secure

Theorem

Assuming that

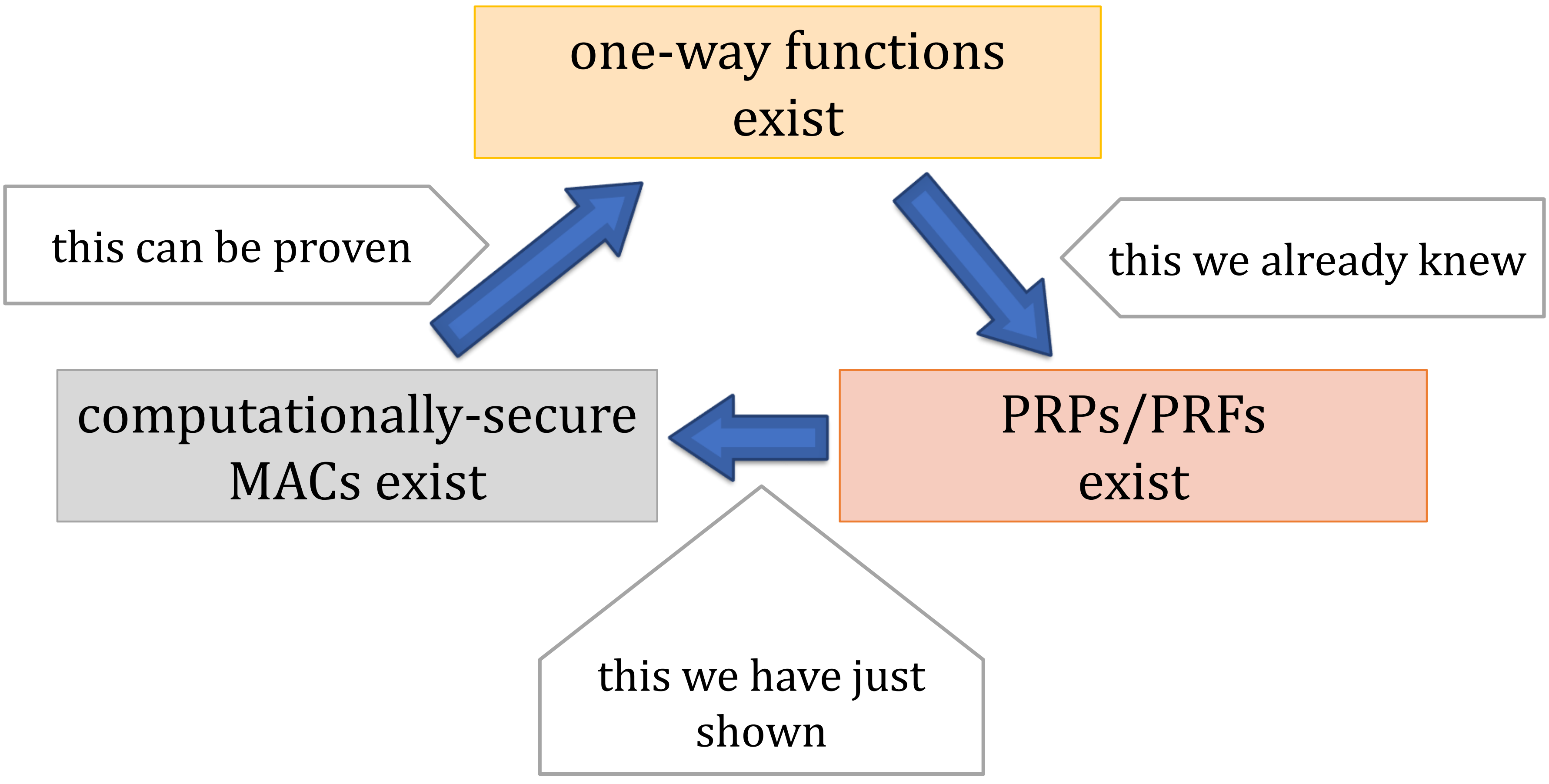
$\boldsymbol{F} : \{\mathbf{0}, \mathbf{1}\}^{\boldsymbol{n}} \times \{\mathbf{0}, \mathbf{1}\}^{\boldsymbol{n}} \rightarrow \{\mathbf{0}, \mathbf{1}\}^{\boldsymbol{n}}$ is a **pseudorandom permutation**

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

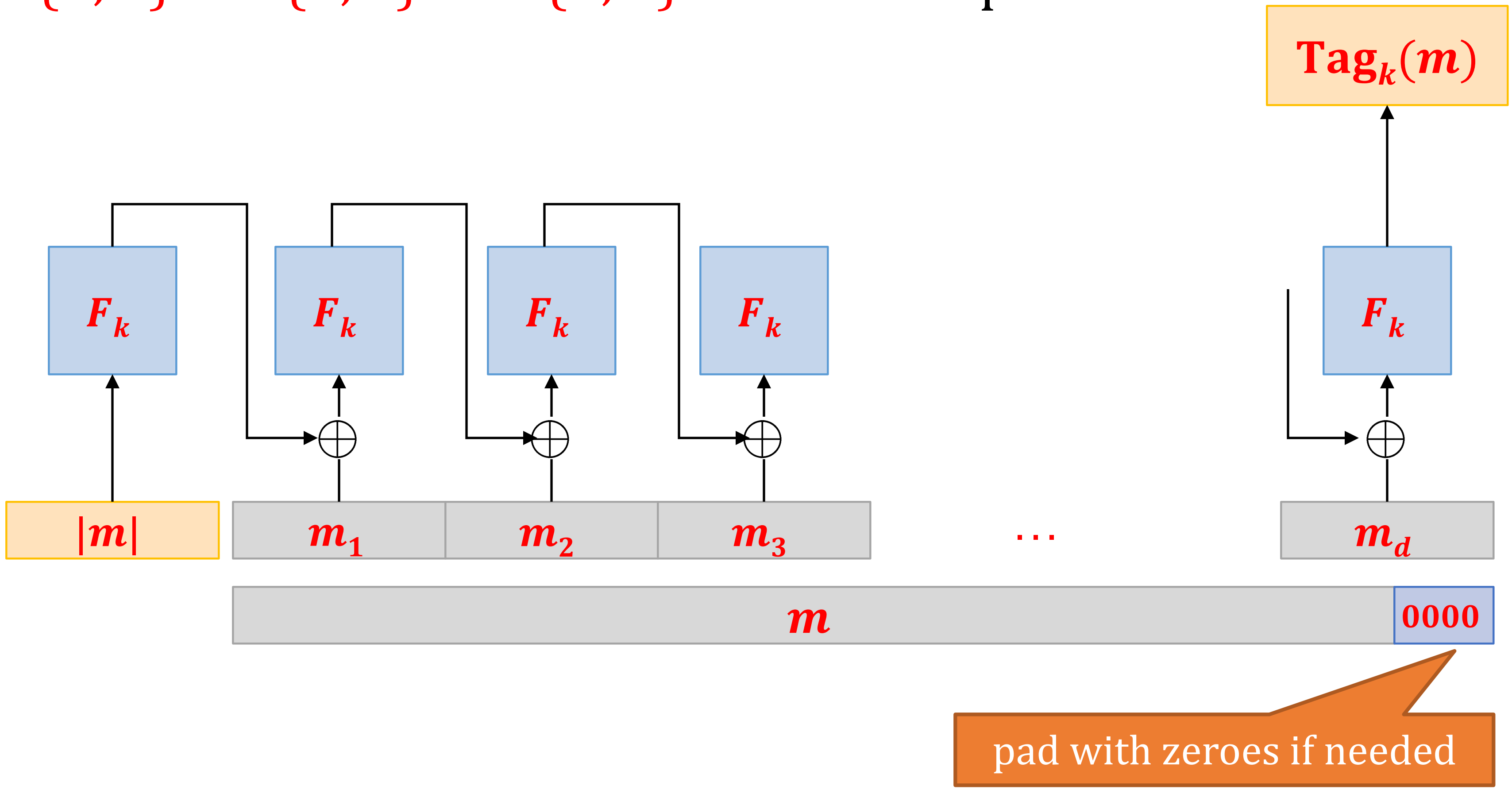
Problem:

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

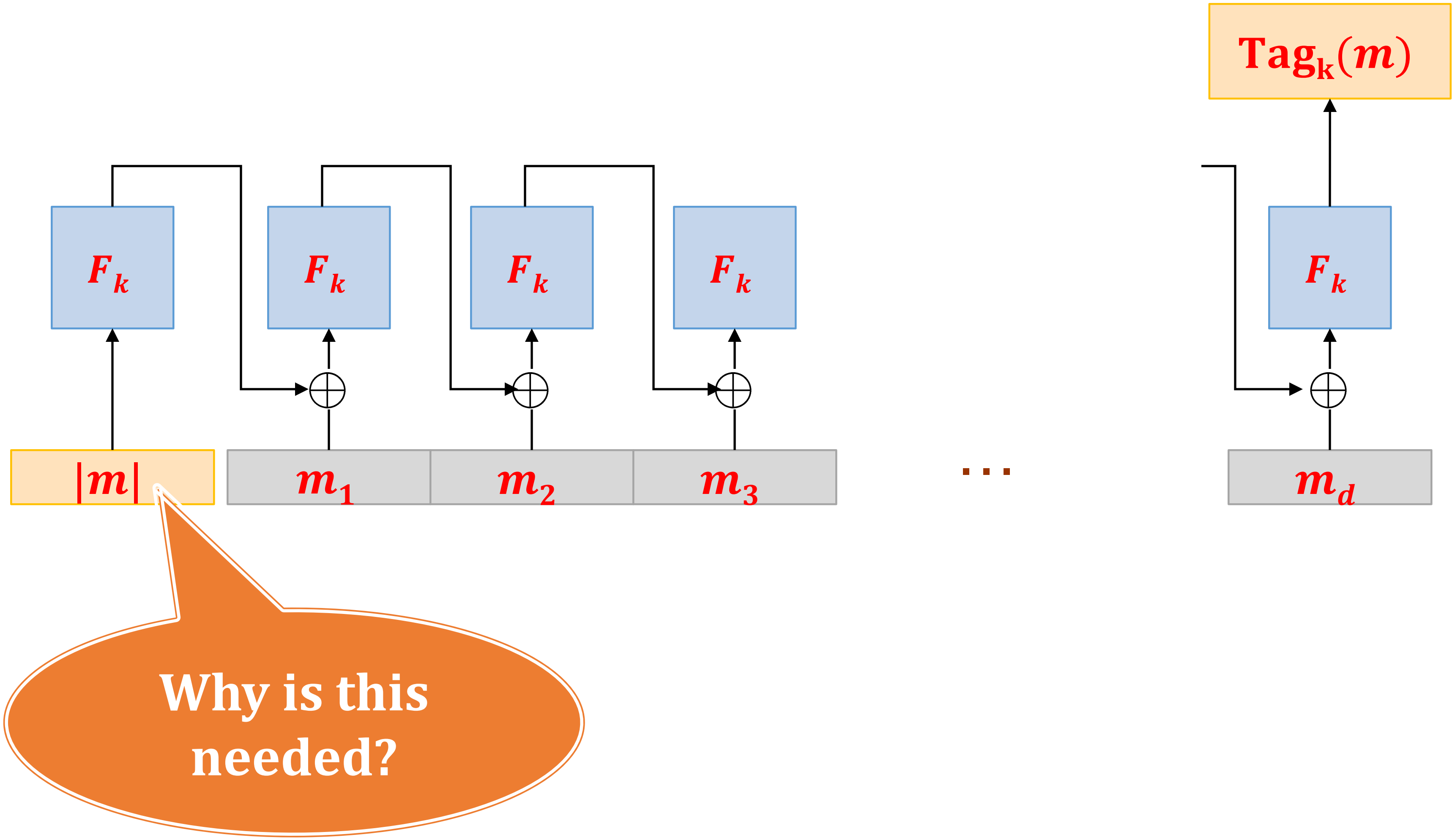
We can do much better!

CBC-MAC

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

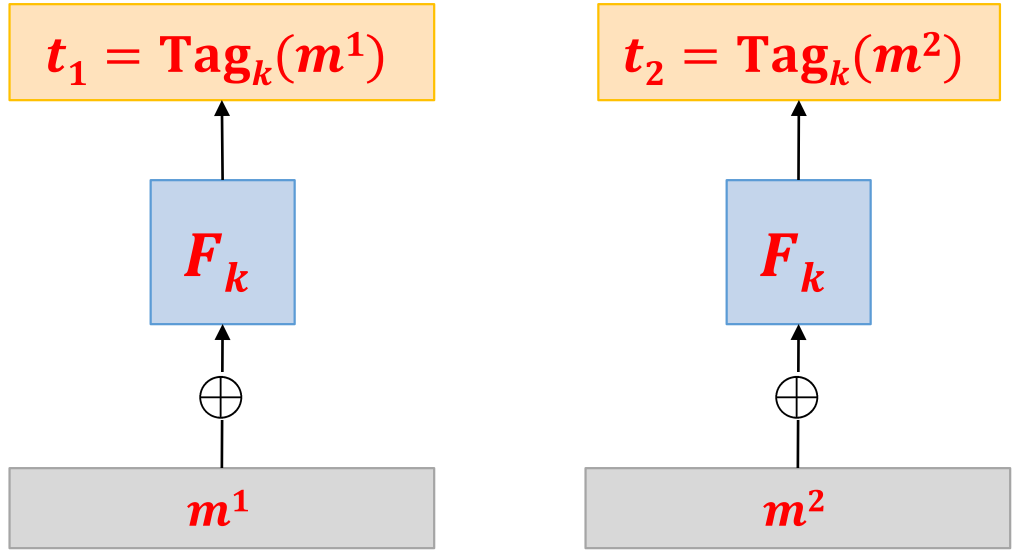


Other variants exist!

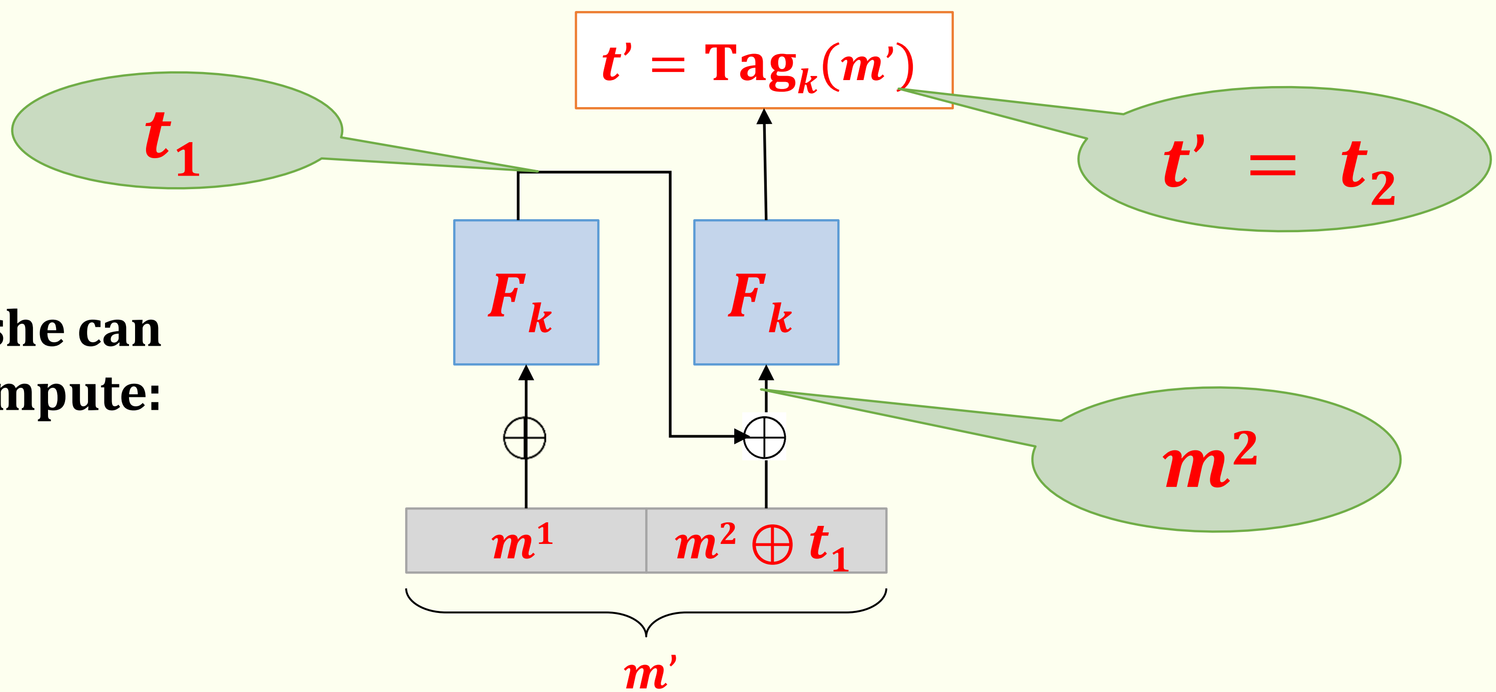


Suppose we do not prepend $|m|$...

the adversary chooses:



now she can compute:



Plan

1. Introduction to Message Authentication Codes (MACs).
2. Constructions of MACs from block ciphers
3. Constructions of MACs from hash functions
4. Authenticated encryption
5. Key establishment with a trusted server
6. Outlook

Some practitioners don't like the CBC-MAC

They prefer to use the **hash functions** for authentication.

Why?

- hash functions tend to be a bit **more efficient**
- **no export regulations** (important in the past)

How to use hash functions for authentication?

A natural idea used by the practitioners:

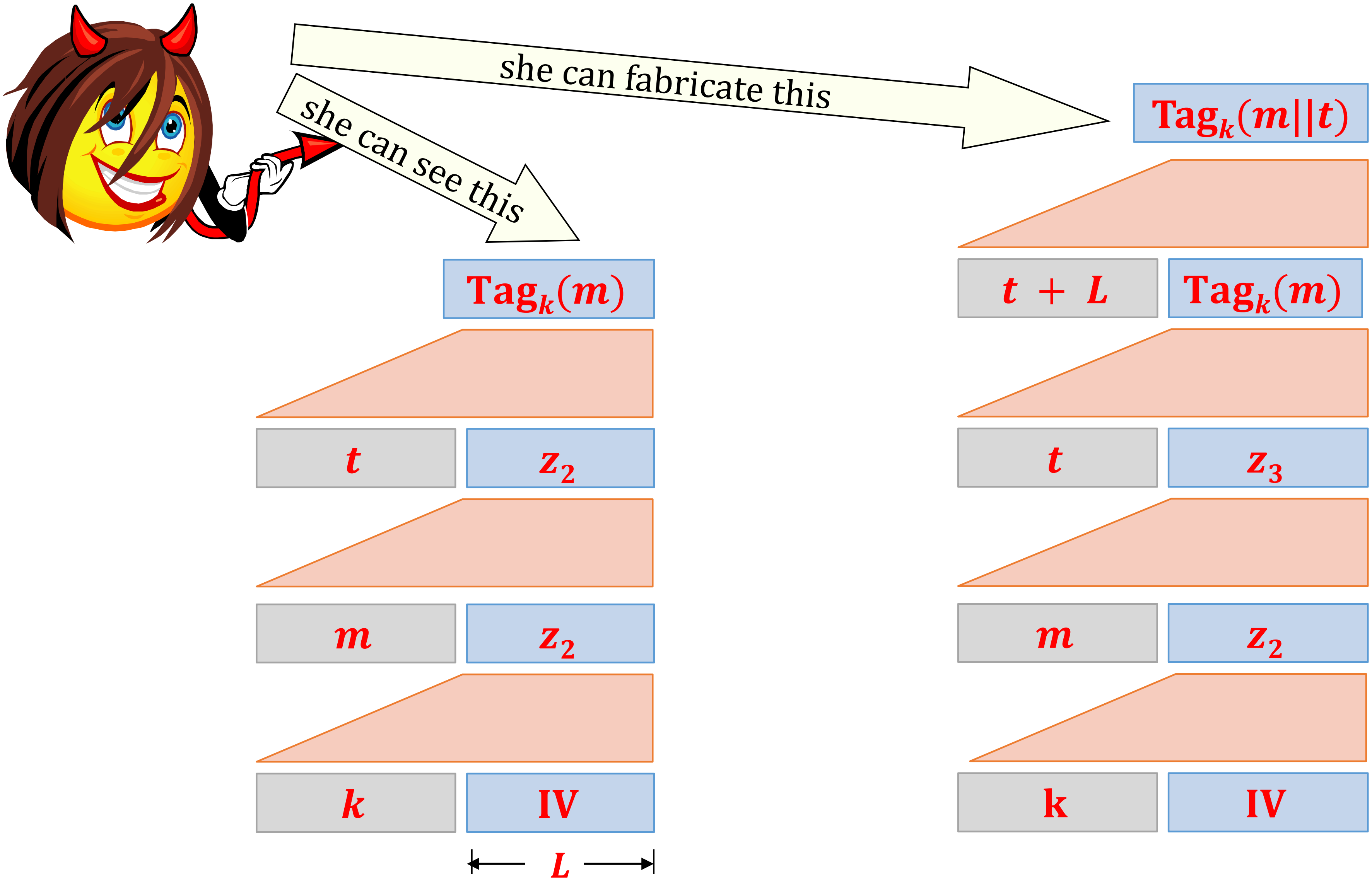
$\boldsymbol{H}$ – hash function

Hash a message together with the key:

$$\mathbf{Tag}_{\boldsymbol{k}}(\boldsymbol{m}) = \boldsymbol{H}(\boldsymbol{k} \,||\, \boldsymbol{m})$$

this is not secure!

Message extension attack: Suppose H was constructed using the **MD-transform**



Still, used in practice in the past

For example in **SSL v.2**:

```
The MAC-DATA is computed as
follows:

MAC-DATA = HASH[SECRET, ACTUAL-
DATA, PADDING-DATA, SEQUENCE-
NUMBER]
```

A better idea

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

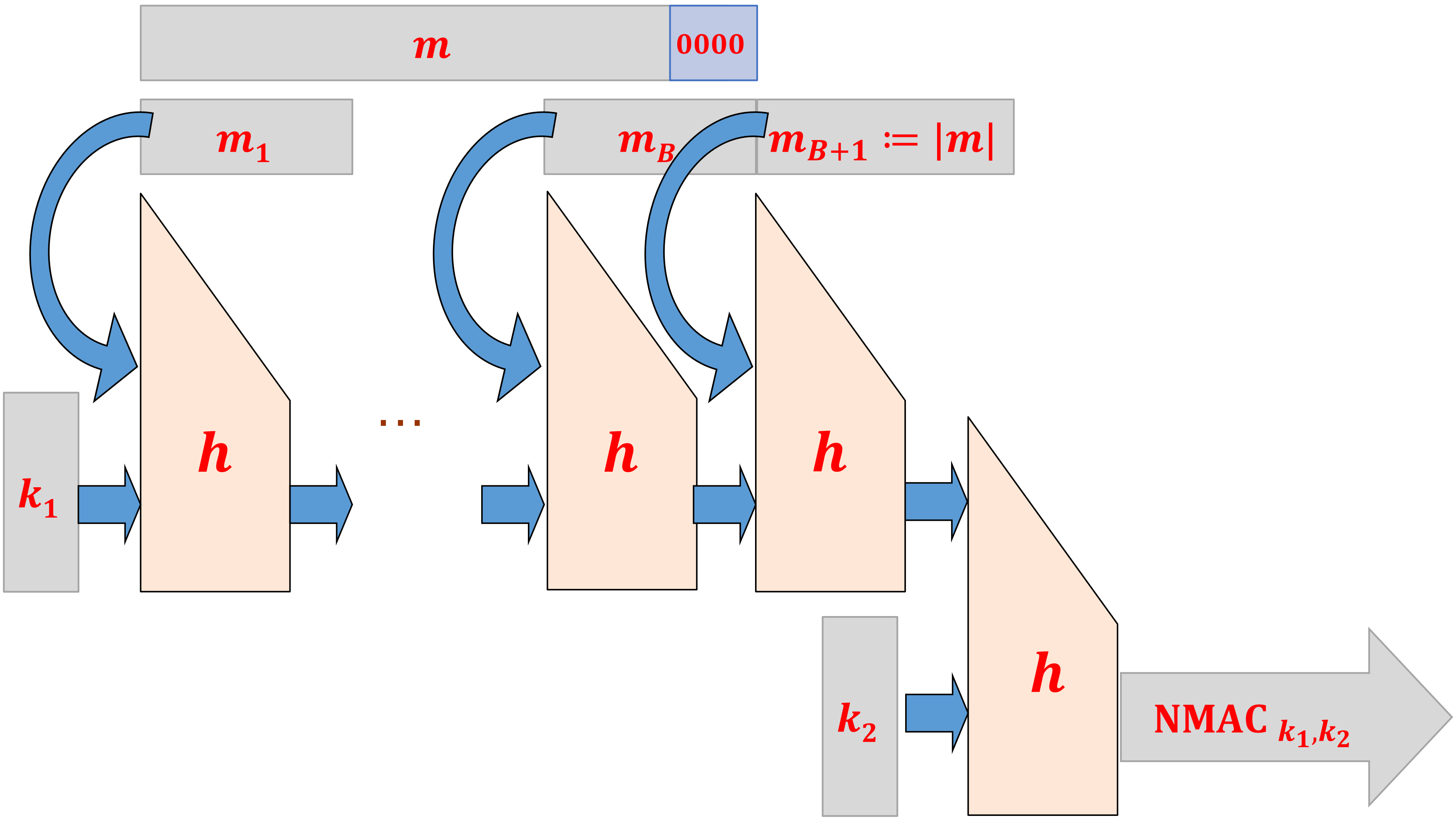
- **NMAC** (Nested MAC)
- **HMAC** (Hash based MAC)

have some “provable properties”

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

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

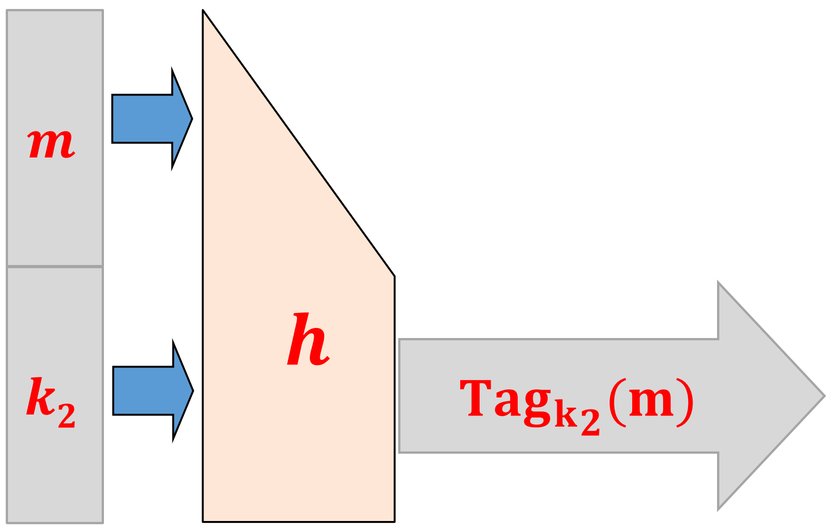
NMAC



What can be proven

Suppose that

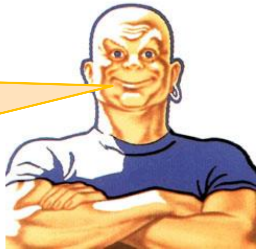
1. $\boldsymbol{h}$ is collision-resistant
2. the following function is a secure **MAC**:



Then NMAC is a secure **MAC**.

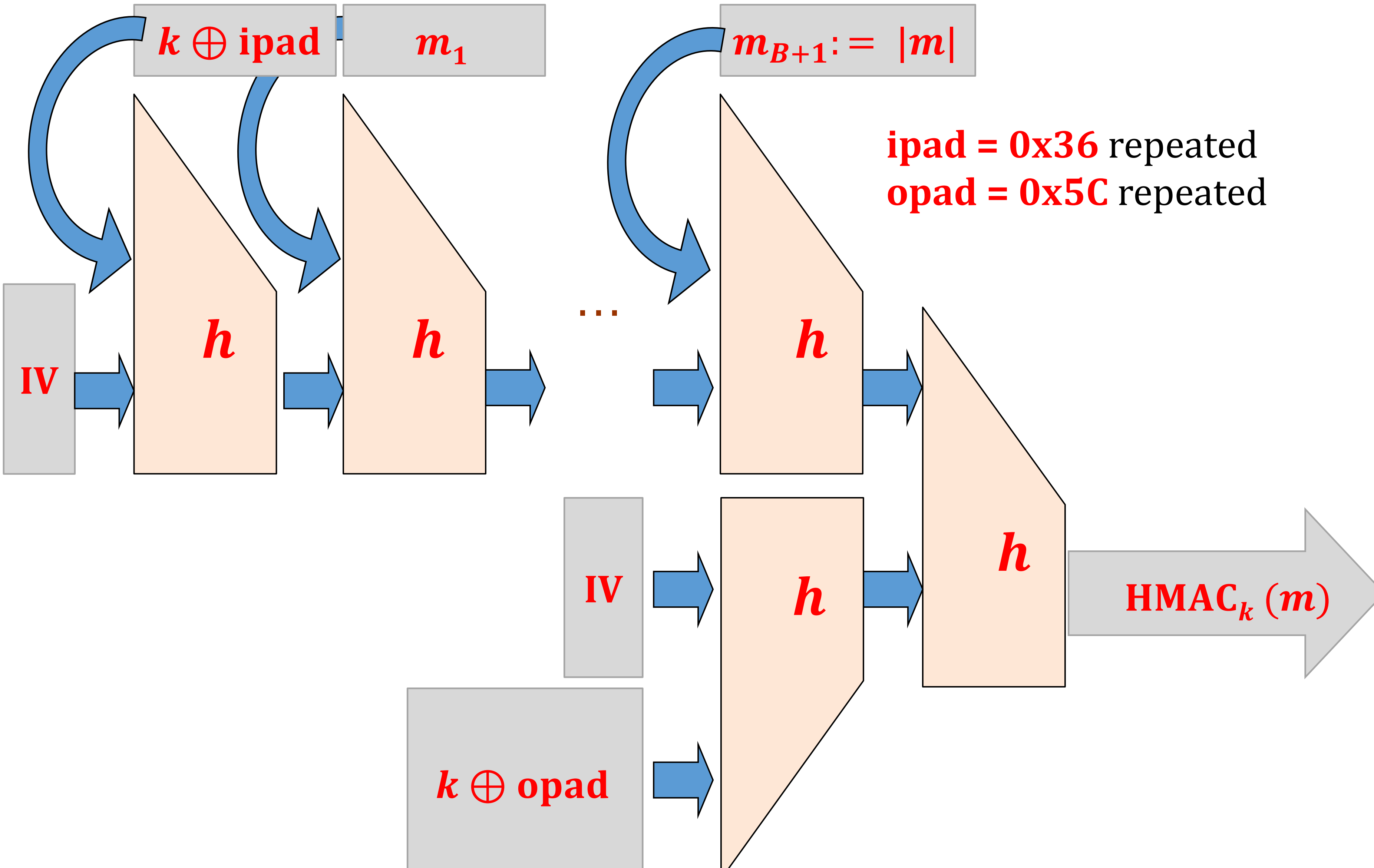
We don't like it:

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



HMAC is the solution!

HMAC



ipad = 0x36 repeated
opad = 0x5C repeated

Why such a choice for **ipad** and **opad**?

in binary:

ipad = 0x36363636…

opad = 0x5C5C5C5C…

0	1	0	1	1	1	0	0	0	1	0	1	1	1	0	0
0	0	1	1	0	1	1	0	0	0	1	1	0	1	1	0

Properties:

- **simple representation** (easier to implement, less error-prone)
- **Hamming distance** between the pads around $\frac{n}{2}$ (where $n = |\mathbf{opad}| = |\mathbf{ipad}|$).

HMAC – the properties

Looks **complicated**, but it is very easy to implement (given an implementation of $\boldsymbol{H}$):

$$\mathbf{HMAC}_{\boldsymbol{k}}(\boldsymbol{m}) \;=\; \boldsymbol{H}((\boldsymbol{k} \oplus \mathbf{opad}) \,||\, \boldsymbol{H}(\boldsymbol{k} \oplus \mathbf{ipad} \,||\, \boldsymbol{m}))$$

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

Widely used in practice.

Plan

1. Introduction to Message Authentication Codes (MACs).
2. Constructions of MACs from block ciphers
3. Constructions of MACs from hash functions
4. Authenticated encryption
5. Key establishment with a trusted server
6. Outlook

What is needed to establish secure channels?

In practice one needs both

encryption

and

authentication.

This can be achieved as follows:

- **combine encryption** with **authentication**

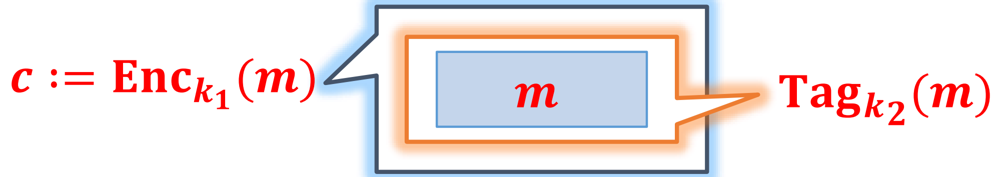
or

- design "**authenticated encryption**" from scratch.

Authentication + encryption, options:

- **Encrypt-and-authenticate:**

$$c := \mathbf{Enc}_{k_1}(m) \text{ and } t := \mathbf{Tag}_{k_2}(m), \text{ send } (c, t)$$



wrong

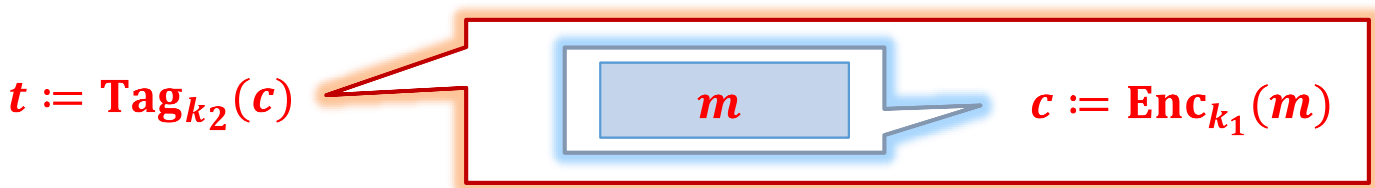
- **Authenticate-then-encrypt:**

$$t := \mathbf{Tag}_{k_2}(m) \text{ and } c := \mathbf{Enc}_{k_1}(m||t), \text{ send } (c, t)$$

better

- **Encrypt-then-authenticate:**

$$c := \mathbf{Enc}_{k_1}(m) \text{ and } t := \mathbf{Tag}_{k_2}(c), \text{ send } (c, t)$$



the best

By the way…

Never use the same key for encryption and authentication.

<u>Actually</u>:

Never use the **same key in two different applications** (or two different instantiations of the same application).

Authenticated encryption

In principle: should be more efficient than the

A popular method: **Galois/Counter Mode**.

An ongoing competition for a new authenticated encryption **scheme**:

> **CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness**
>
> (expected outcome: **2017**)
>
> not formally organized by any institution, supported by a grant from NIST
> **webpage:** competitions.cr.yp.to/caesar-submissions.html

Caesar competition **third-round** candidates

candidate	designers
ACORN	Hongjun Wu
AEGIS	Hongjun Wu, Bart Preneel
AES-OTR	Kazuhiko Minematsu
AEZ	Viet Tung Hoang, Ted Krovetz, Phillip Rogaway
Ascon	Christoph Dobraunig, Maria Eichlseder, Florian Mendel, Martin Schläffer
CLOC and **SILC**	Tetsu Iwata, Kazuhiko Minematsu, Jian Guo, Sumio Morioka, Eita Kobayashi
COLM	Elena Andreeva, Andrey Bogdanov, Nilanjan Datta, Atul Luykx, Bart Mennink, Mridul Nandi, Elmar Tischhauser, Kan Yasuda

candidate	designers
Deoxys	Jérémy Jean, Ivica Nikolić, Thomas Peyrin, Yannick Seurin
JAMBU	Hongjun Wu, Tao Huang
Ketje	Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche, Ronny Van Keer
Keyak	Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche, Ronny Van Keer
MORUS	Hongjun Wu, Tao Huang
NORX	Jean-Philippe Aumasson, Philipp Jovanovic, Samuel Neves
OCB	Ted Krovetz, Phillip Rogaway
Tiaoxin	Ivica Nikolić

Plan

1. Introduction to Message Authentication Codes (MACs).
2. Constructions of MACs from block ciphers
3. Constructions of MACs from hash functions
4. Authenticated encryption
5. Key establishment with a trusted server
6. Outlook

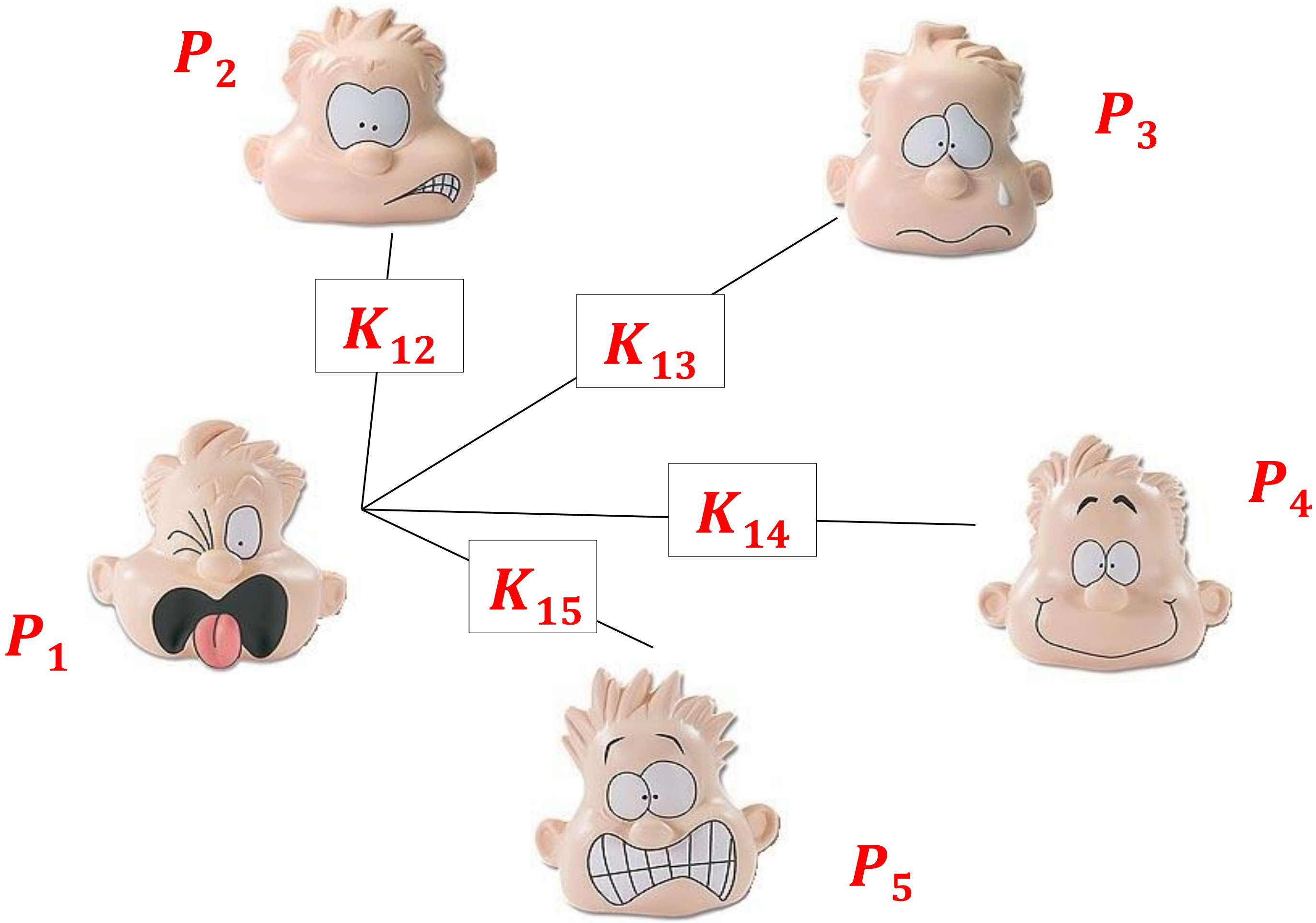
How to distribute the cryptographic keys?

If the users can meet in person beforehand – **it's simple.**

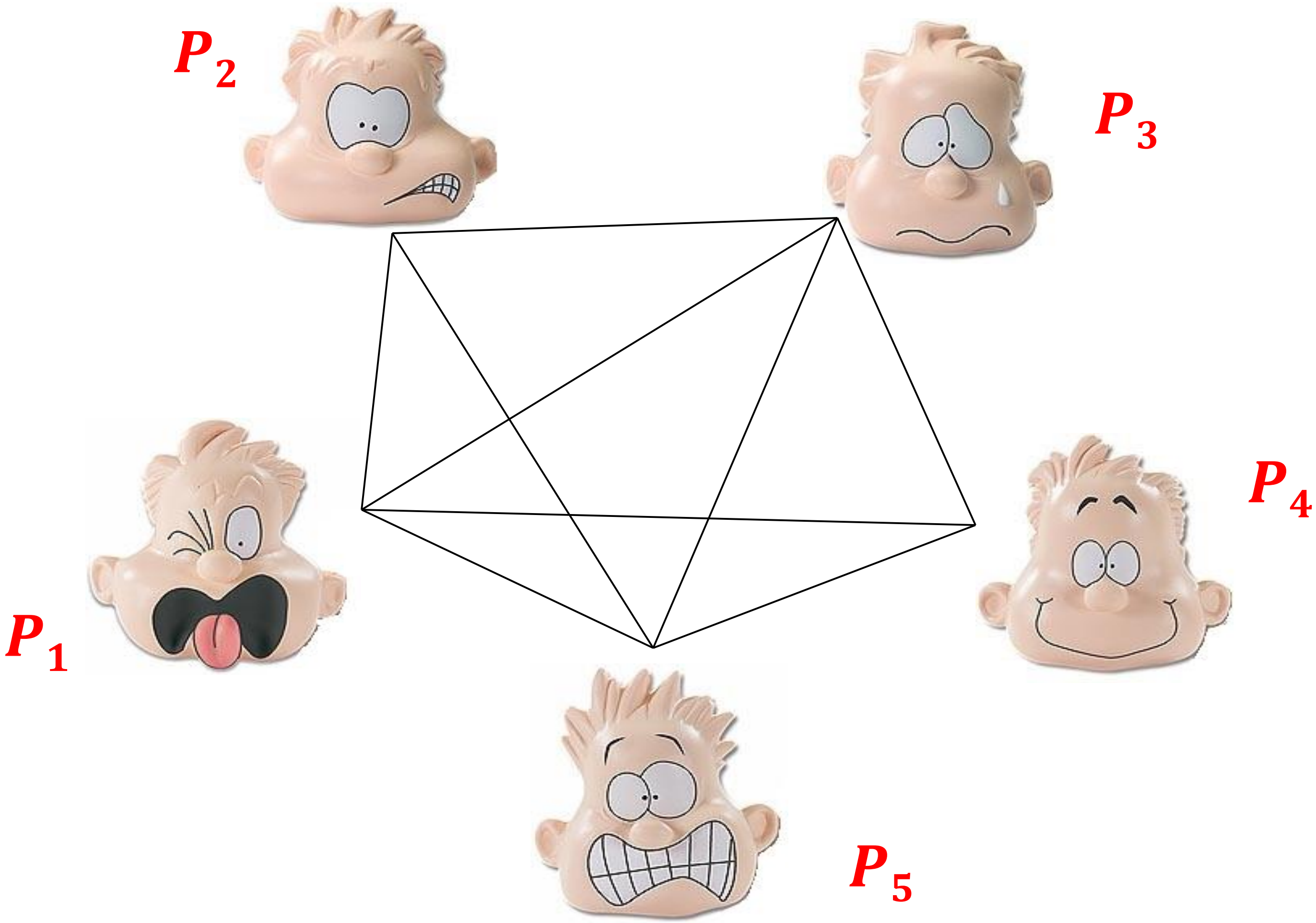
But what to do if they **cannot meet**?

(example: **on-line shopping**)

A naive solution: give to every user P_i a separate key K_{ij} to communicate with every P_j



In general: a **quadratic** number of keys is needed



Key Distribution Centers

Some **server** (a **Key Distribution Center, KDC**) "gives the keys" to the users

- **feasible** if the users are e.g. working in one company
- **infeasible** on the internet
- relies on the honesty of **KDC**
- **KDC** needs to be permanently available
- ...

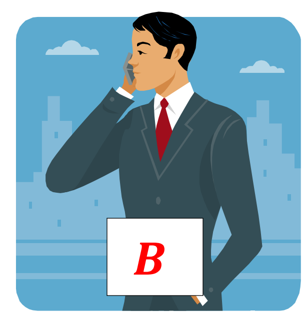
How to establish a key with a trusted server?

key shared by **Alice** and the **server**: K_{AS}

server S

key shared by **Bob** and the **server**: K_{BS}

want to establish a **fresh session key**



Not so trivial as it may seem!

Notation

a message $\boldsymbol{M}$ encrypted and authenticated with $\boldsymbol{K}$:

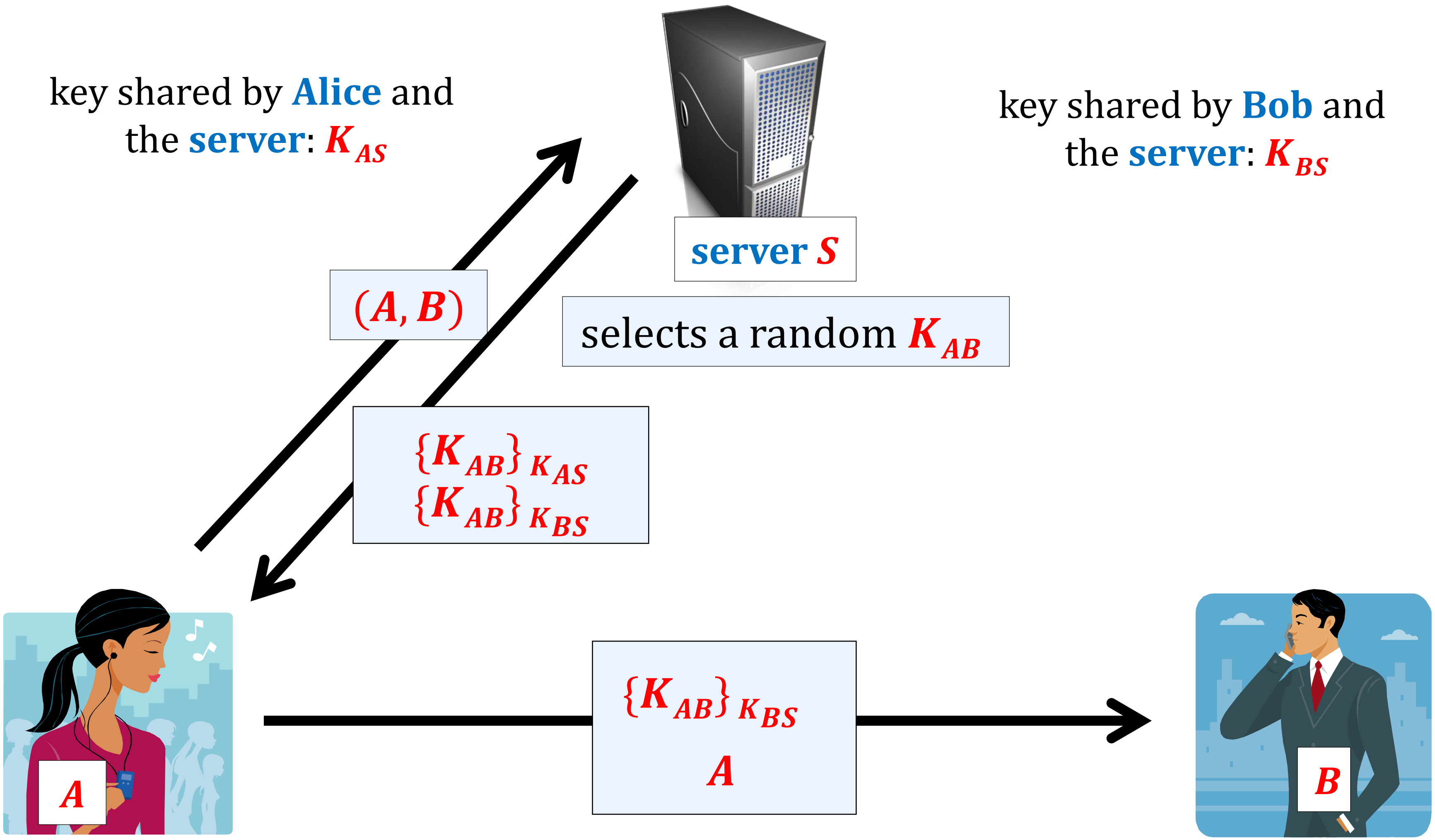
$$\{\boldsymbol{M}\}_{\boldsymbol{K}}$$

Formally:

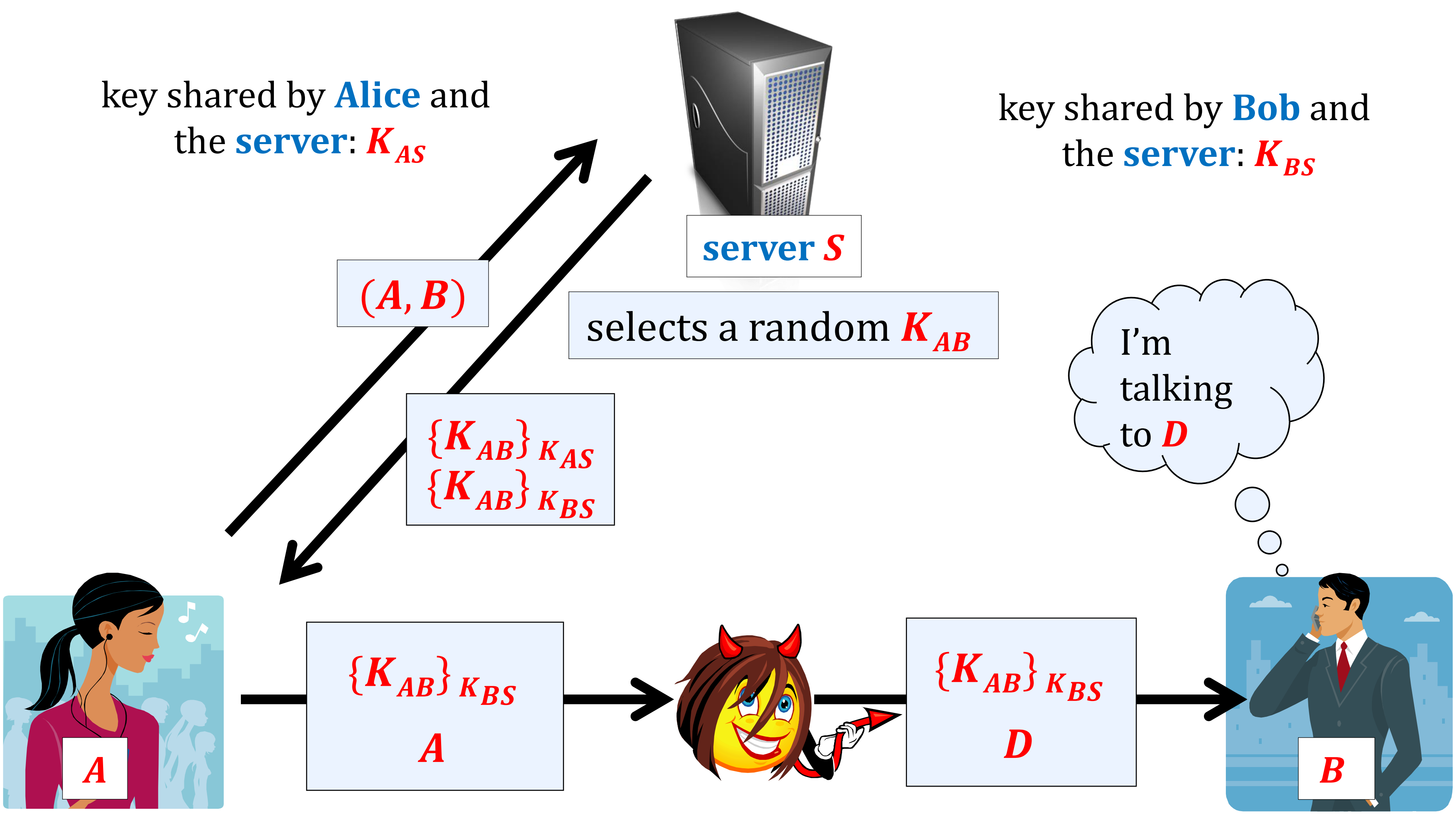
$$\boldsymbol{K} = (\boldsymbol{K}_0, \boldsymbol{K}_1)$$

$$\{\boldsymbol{M}\}_{\boldsymbol{K}} = (\mathbf{Tag}_{\boldsymbol{K}_0}(\mathbf{Enc}_{\boldsymbol{K}_1}(\boldsymbol{M})), \mathbf{Enc}_{\boldsymbol{K}_1}(\boldsymbol{M}))$$

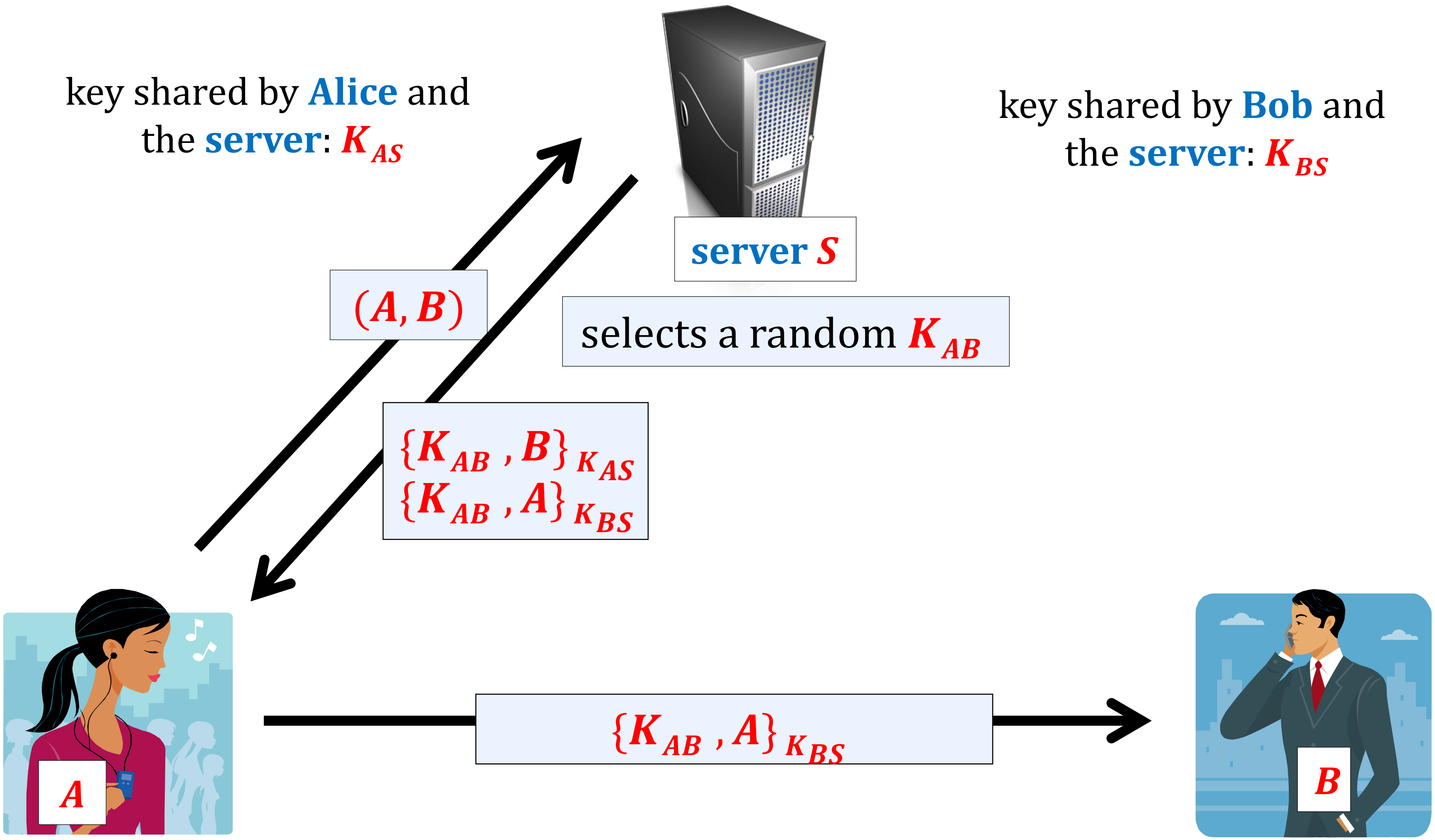
An idea (1)


key shared by Alice and the server: K_{AS}
key shared by Bob and the server: K_{BS}
server S
(A, B)
selects a random K_{AB}
$\{K_{AB}\}_{K_{AS}}$
$\{K_{AB}\}_{K_{BS}}$
$\{K_{AB}\}_{K_{BS}}$
A
A
B

An attack

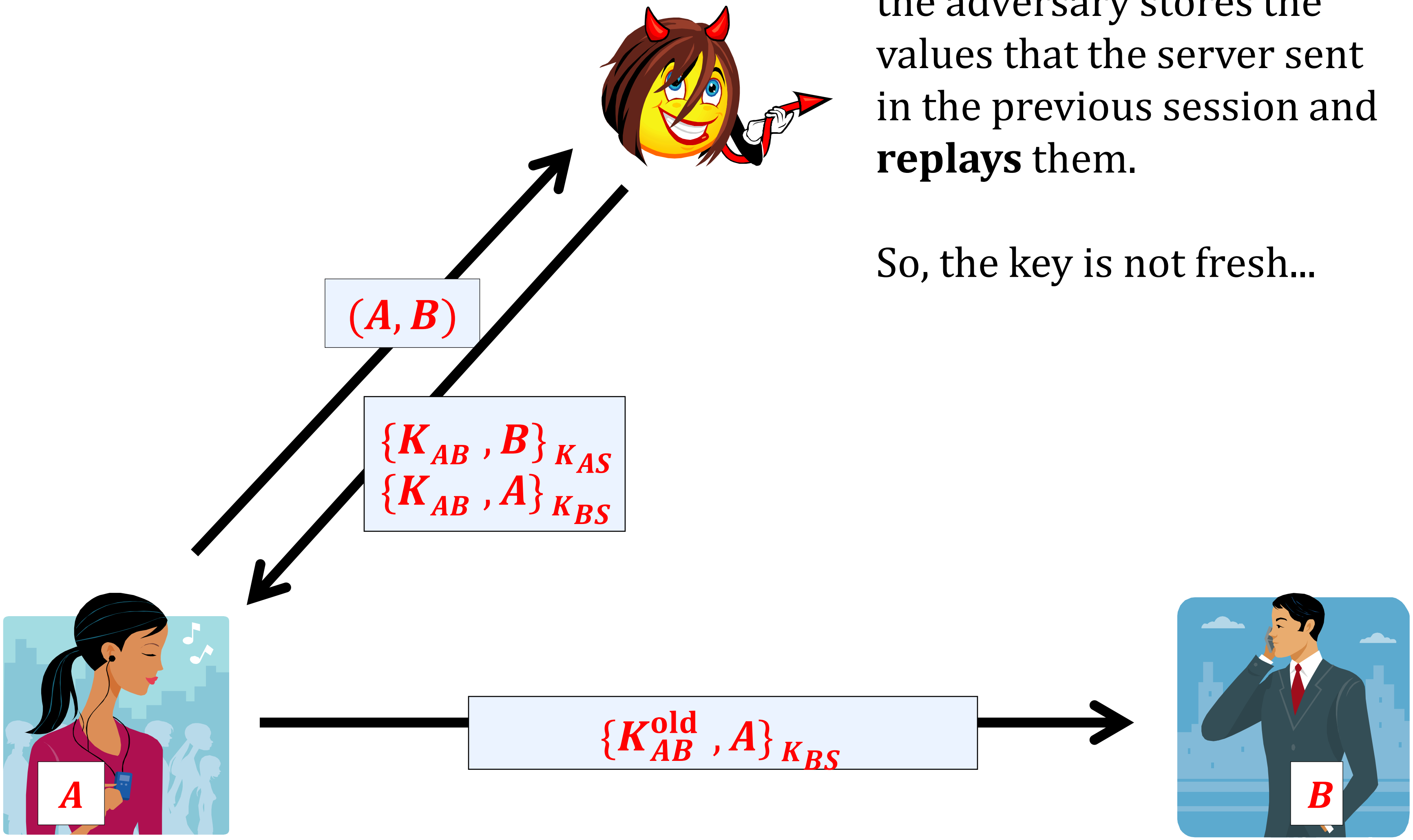

key shared by Alice and the server: K_{AS}
key shared by Bob and the server: K_{BS}
server S
(A, B)
selects a random K_{AB}
$\{K_{AB}\}_{K_{AS}}$
$\{K_{AB}\}_{K_{BS}}$
I'm talking to D
A
$\{K_{AB}\}_{K_{BS}}$
A
$\{K_{AB}\}_{K_{BS}}$
D
B

An idea (2)



key shared by Alice and the server: K_{AS}
key shared by Bob and the server: K_{BS}
server S
(A, B)
selects a random K_{AB}
$\{K_{AB}, B\}_{K_{AS}}$
$\{K_{AB}, A\}_{K_{BS}}$
$\{K_{AB}, A\}_{K_{BS}}$
A
B

A replay attack



the adversary stores the values that the server sent in the previous session and **replays** them.

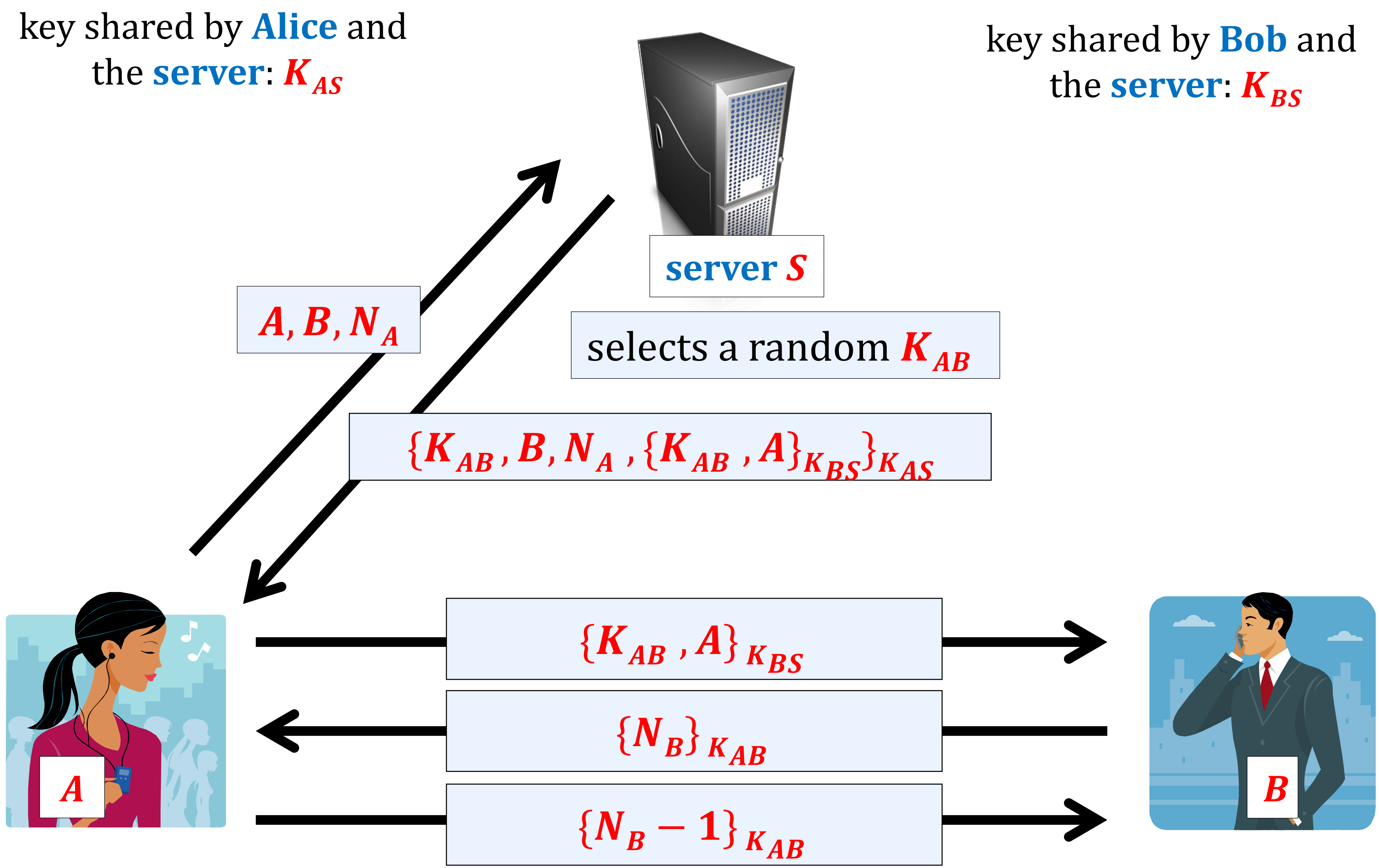
So, the key is not fresh...

How to protect against the replay attacks?

Nonce – “number used once”.

Nonce will be generated by one party and returned to that party to show that a **message is newly generated**.

An idea (3): Needham Schreoder 1972.



An attack on Needham Schroeder

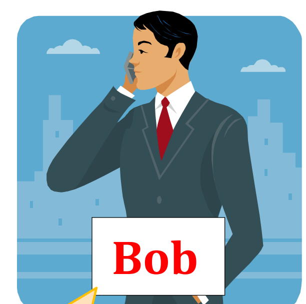
Assume that an old session key K_{AB}^{old} is known to the adversary.

For example if K_{AB}^{old} is used as one-time pad this may happen...

$\{K_{AB}^{\mathrm{old}}, A\}_{K_{BS}}$ $\rightarrow$

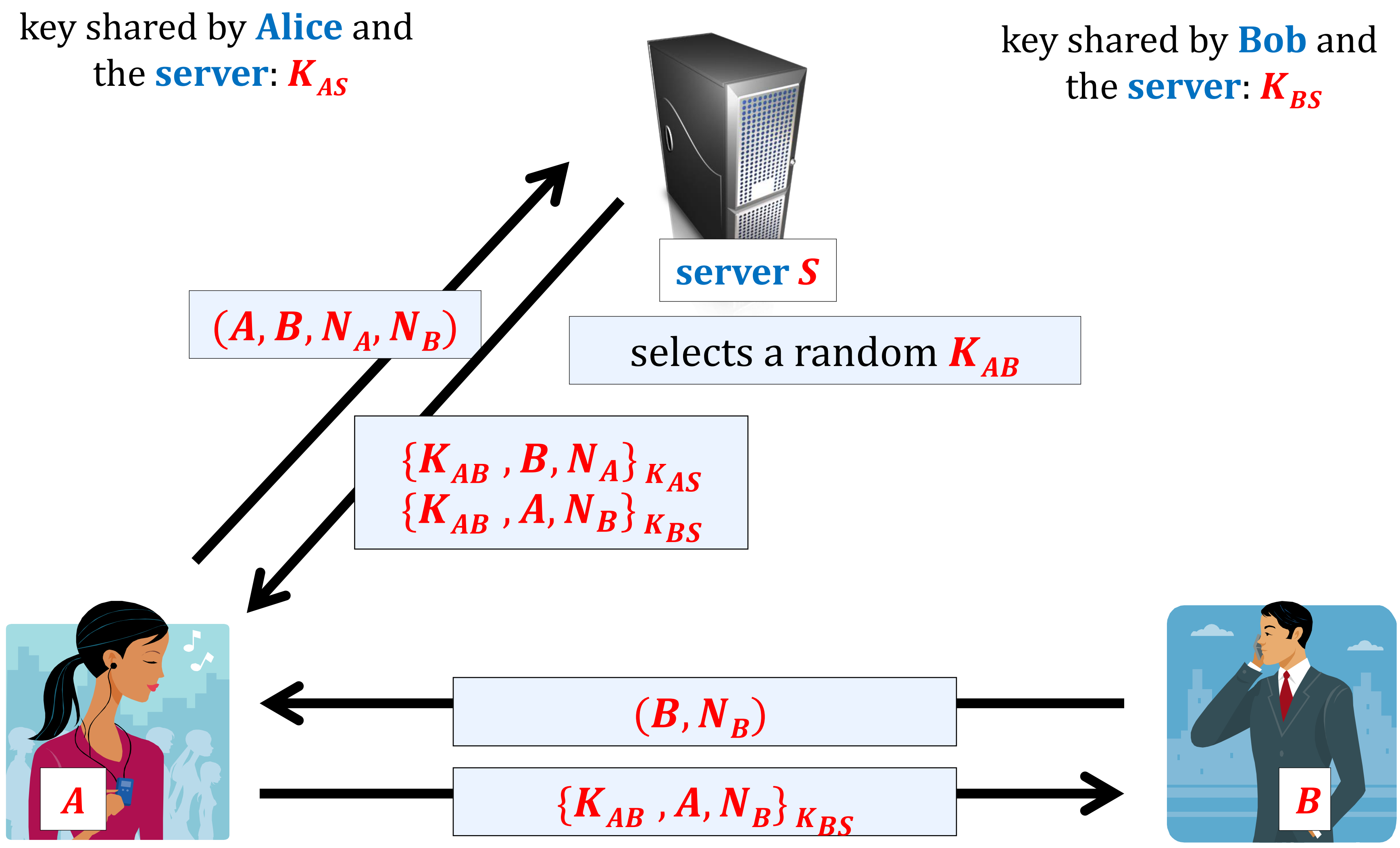
$\leftarrow$ $\{N_B\}_{K_{AB}^{\mathrm{old}}}$

$\{N_B - 1\}_{K_{AB}^{\mathrm{old}}}$ $\rightarrow$



Bob will accept K_{AB}^{old} as the new key

The final solution



How it looks in practice

Some systems that are based on trusted server have been used in practice (e.g. **Kerberos**).

One major problem: why shall we **trust the server**?

Solution:

use the **public-key cryptography**

(next lecture)

Plan

1. Introduction to Message Authentication Codes (MACs).
2. Constructions of MACs from block ciphers
3. Constructions of MACs from hash functions
4. Authenticated encryption
5. Key establishment with a trusted server
6. Outlook

Outlook

cryptography

“information-theoretic”, “unconditional”

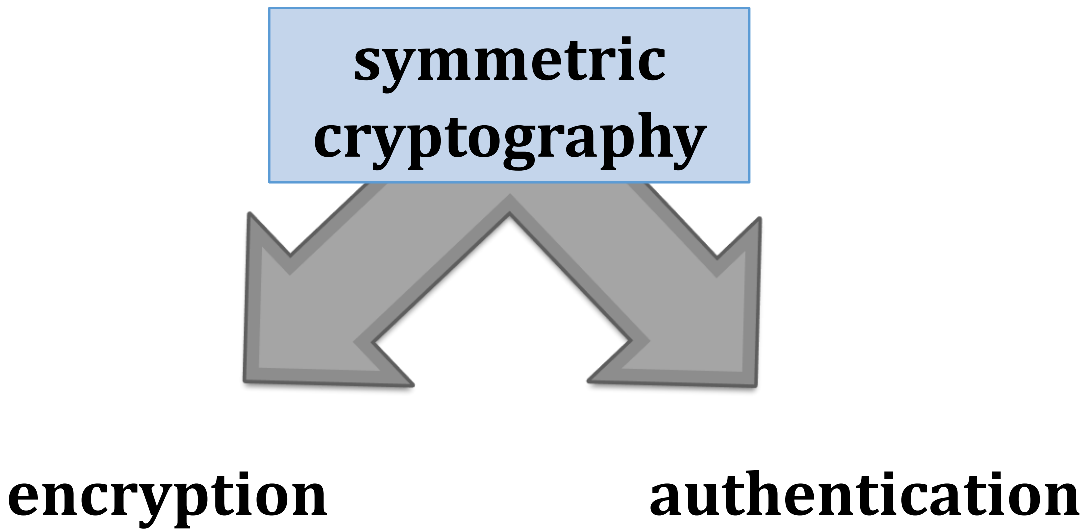
- one time pad,
- quantum cryptography,
- ...

“computational”

based on **2** simultanious assumptions:

1. some problems are computationally difficult
2. our understanding of what “computational difficulty” means is correct.

Symmetric cryptography



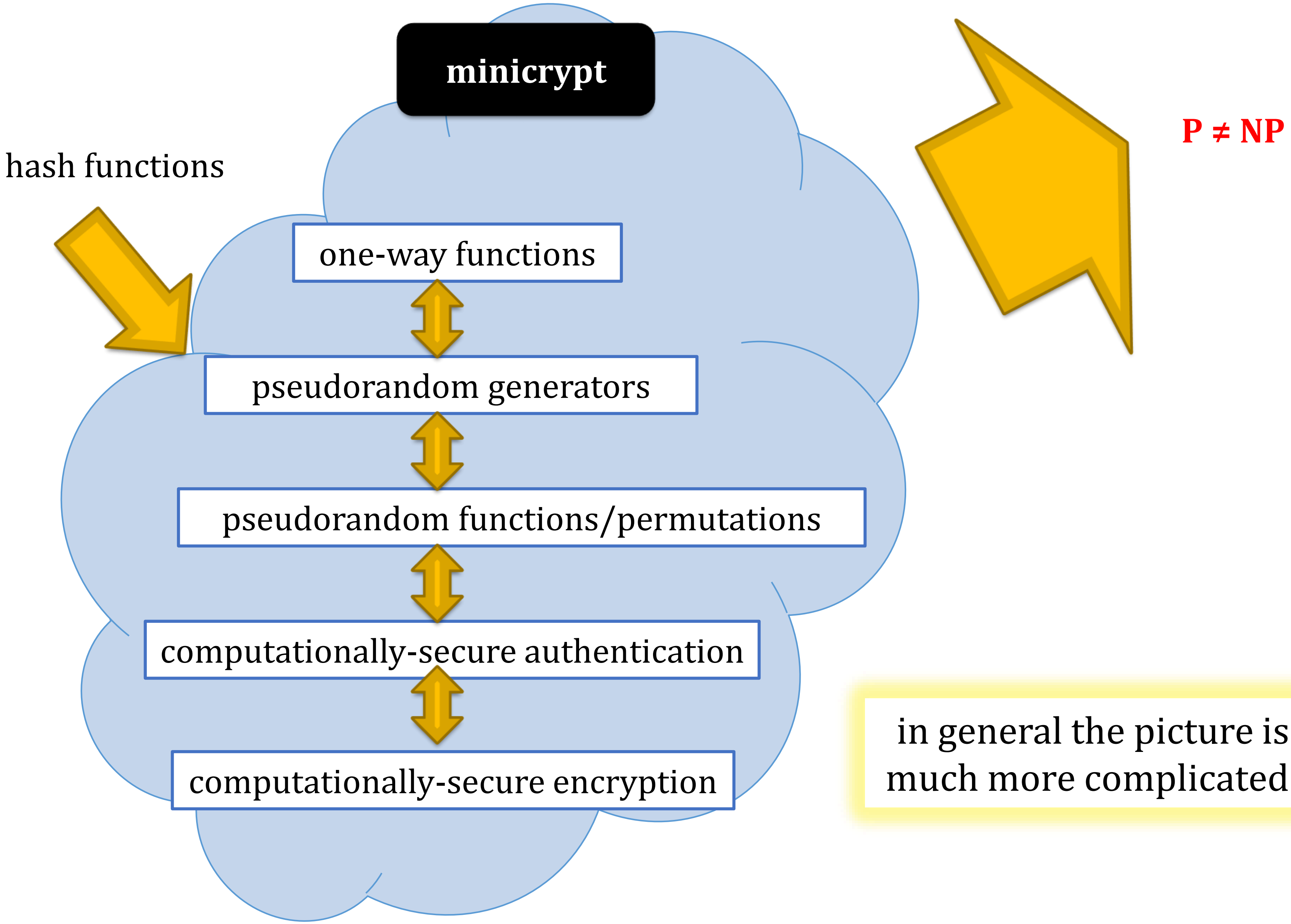
The basic information-theoretic tool

xor (one-time pad)

Basic tools from the computational cryptography

- **one-way functions**
- **pseudorandom generators**
- **pseudorandom functions/permutations**
- **hash functions**

A method for proving security: **reductions**



in general the picture is much more complicated!