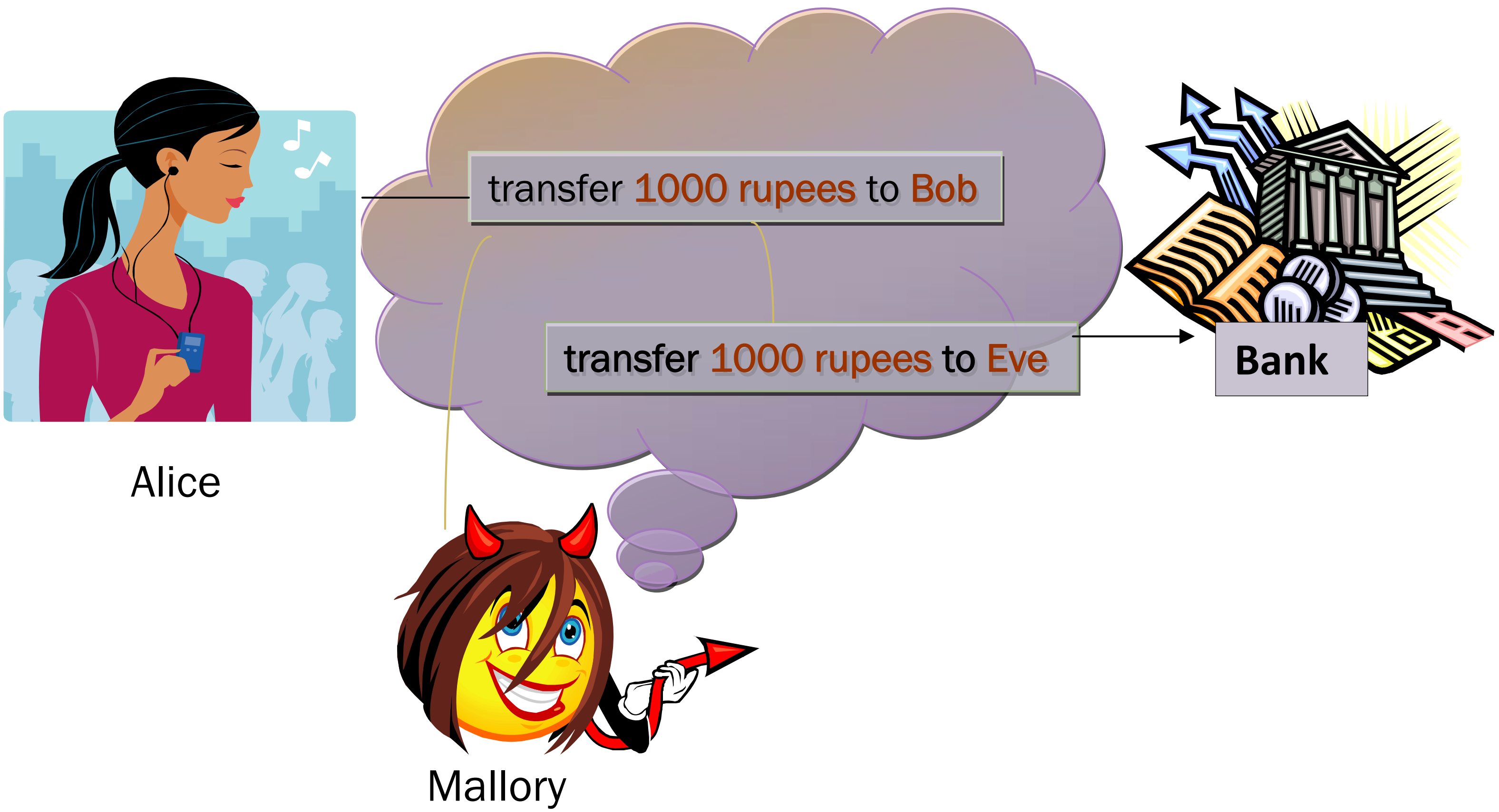
Message Authentication Codes

Need, Construction and Attacks

Problem of Data Integrity



How to prevent such a modification of data?

Does Encryption Guarantee Integrity?

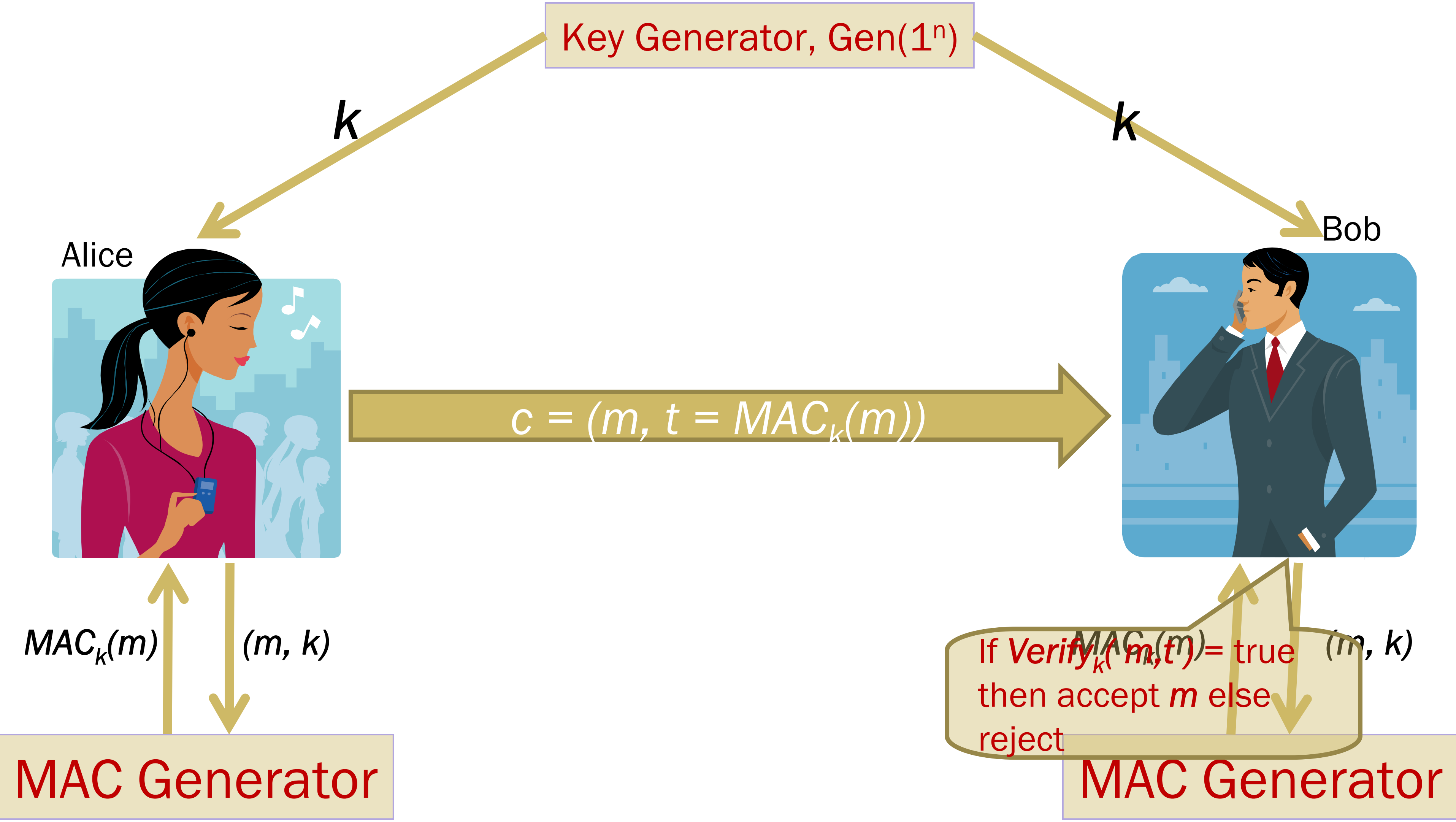
Answer: NO!

For example, consider the encryption using stream ciphers (PRG)

- $c = G(k) \oplus m$
- ciphertext can be manipulated and plaintext is correspondingly modified !

As long as almost all ciphertexts corresponds to some valid plaintext, it is easy for the adversary to “spoof” it

Data Authentication using a MAC

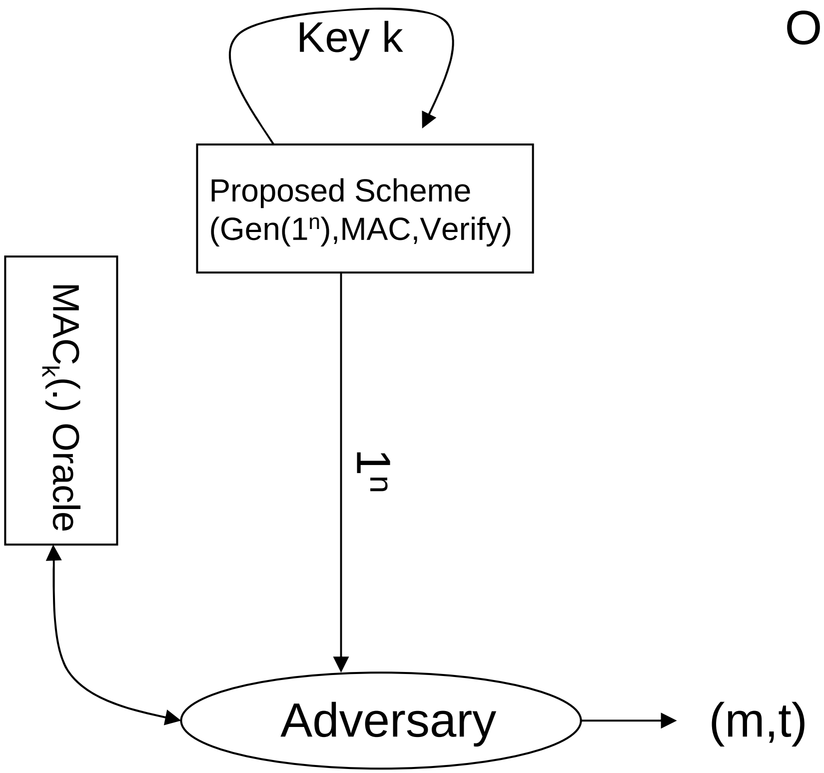


Components of the Authentication Protocol

- A Key Generation Algorithm that returns a secret key *k*
- A MAC generating algorithm that returns a tag for a given message *m*. Tag $t = MAC_k(m)$
- A Verification algorithm that returns a bit $b = Verify_k(m_1, t_1)$, given a message m_1 and a tag t_1
- If the message is not modified then with high probability, the value of *b* is true otherwise false

Security of MAC

Mac-Game(n)



Let Q be the set of all queries from Adv to oracle

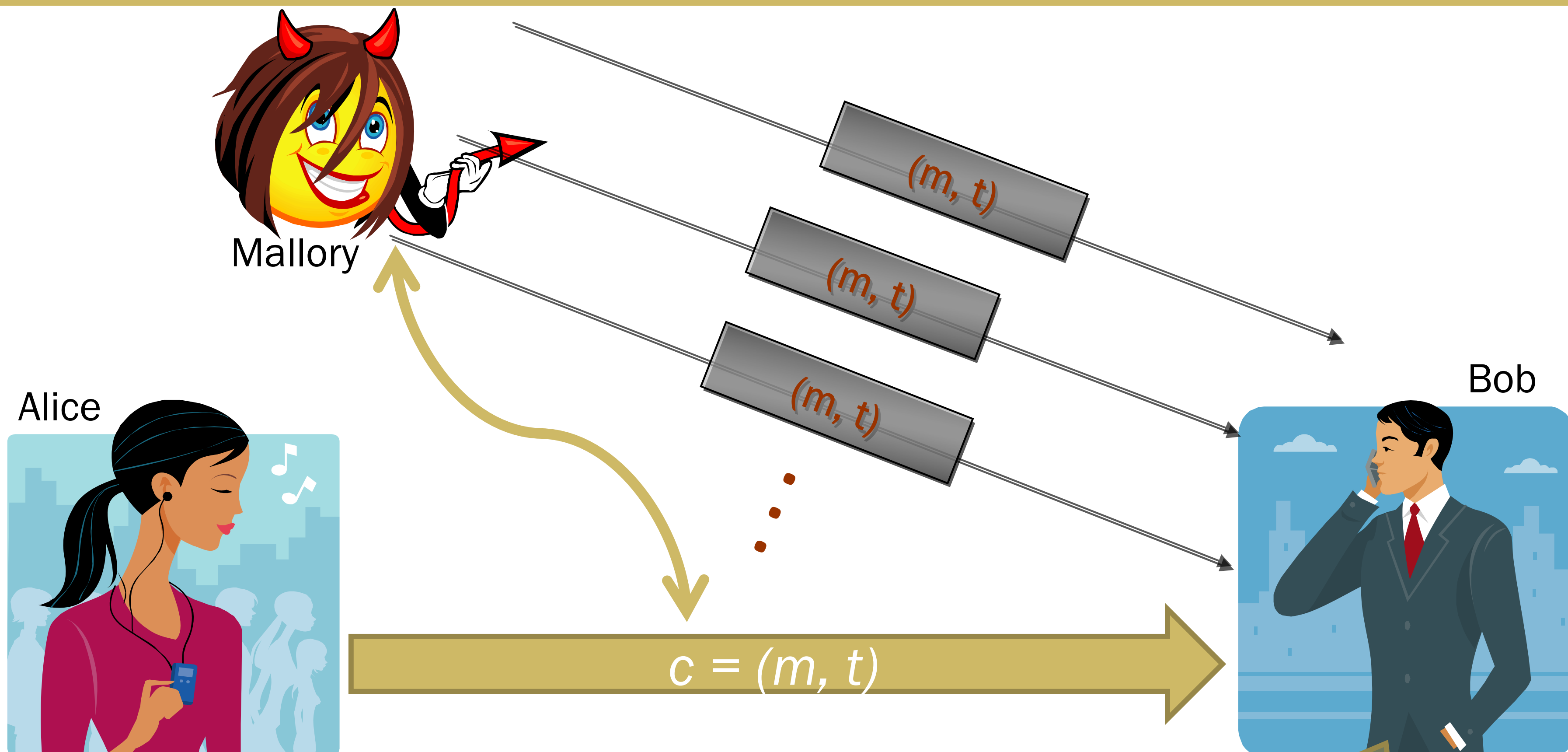
Output of the Game is 1 if and only if:

$\text{Verify}_k(m,t) = 1$ and m is not in Q

A message authentication code (Gen,MAC,Verify) is secure if for all probabilistic polynomial-time adversaries *A*, there exists a negligible function negl such that

$$\Pr[\text{Mac-Game}(n) = 1] \leq \text{negl}(n)$$

Replay Attack



Verify has no "*memory*", cannot detect that ***(m, t)*** is not fresh!

How to solve this problem?

Sequence numbers/timestamps can be used

Construction of MAC using a PRF

Gen **(1^n)** chooses k to be a random n-bit string

MAC_k $(m) = F_k$ $(m) = t$ (the tag)

$Verify_k$ (m, t) = Accept, if and only if ***$t = F_k$ (m)***

Theorem: If F is a pseudorandom function, the above scheme is a secure *fixed length* **MAC**

Variable Length MACs (Method 1)

- Partition the message *m* to *n* sized blocks $m_1 m_2 \ldots m_q$
- Calculate $MAC_k(m) = MAC_k(m_1 \oplus m_2 \ \ldots \oplus m_q)$
- Is this method Secure?

- NO! We are authenticating the *xor* of the message blocks but not the message itself. So we can always choose a message whose *xor* value is the same as some other message

Variable Length MACs (Method 2)

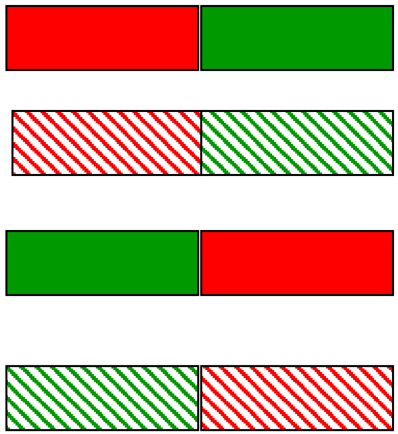
- Concatenate the TAG values of all blocks calculated separately
- But the adversary can rearrange the message blocks and respective tags generating the new message and tag

m:

$t = MAC_k(m)$:

m’ = permutation(m):

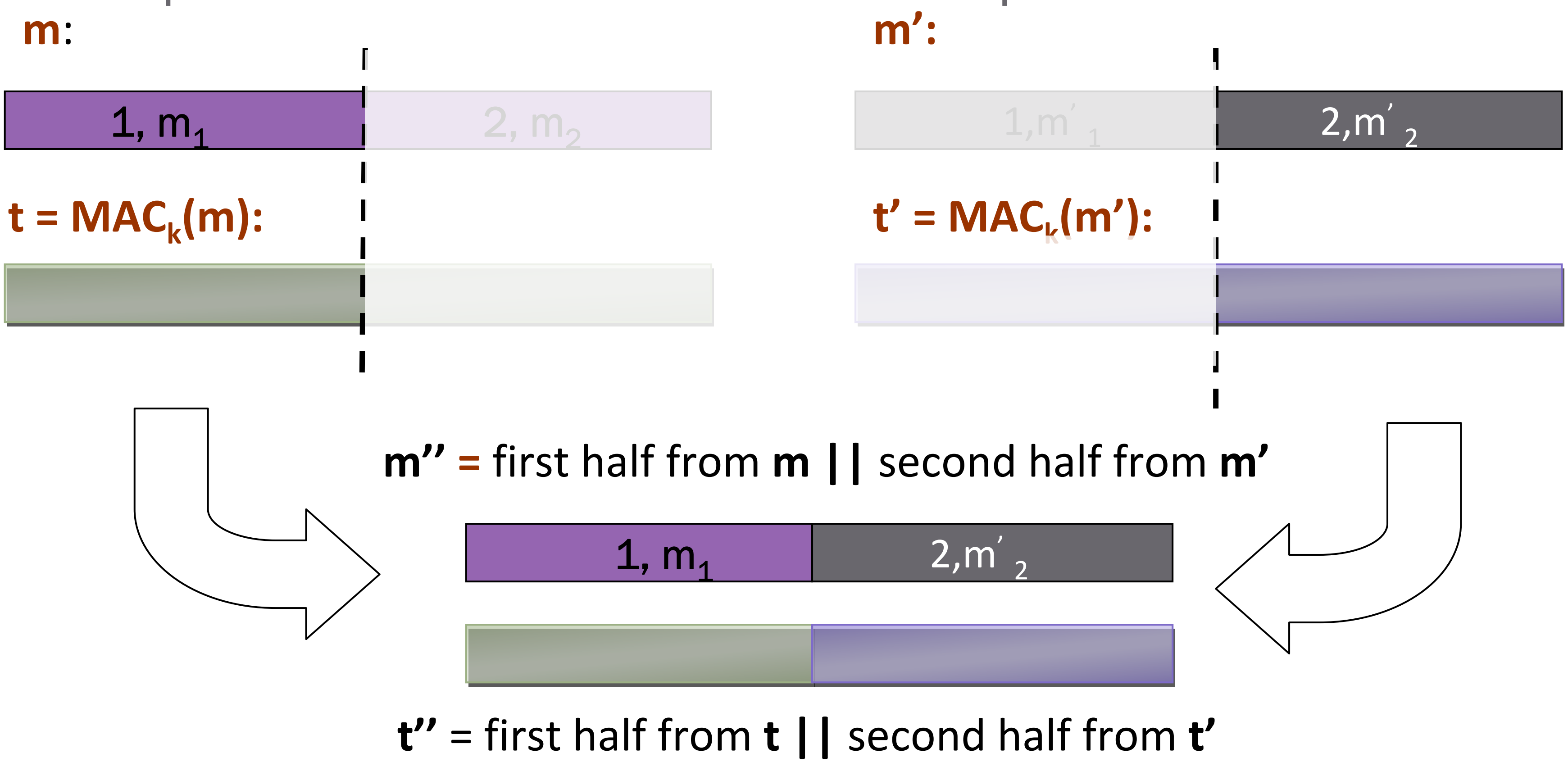
t’ = permutation(t):



- Not Secure!

Variable Length MACs (Method 3)

- To prevent the reordering in previous method, we use sequence numbers. But consider the problem below:



- Then **t''** is a valid tag on **m''**. Not Secure!

Variable Length MACs (Method 4)

- To prevent the above attack we need to keep track of previous message's last sequence number and continue the sequence. So, we send it as

m:

$1, m_1$	$2, m_2$

m':

$3, m'_1$	$4, m'_2$

t = MAC$_k$(m):

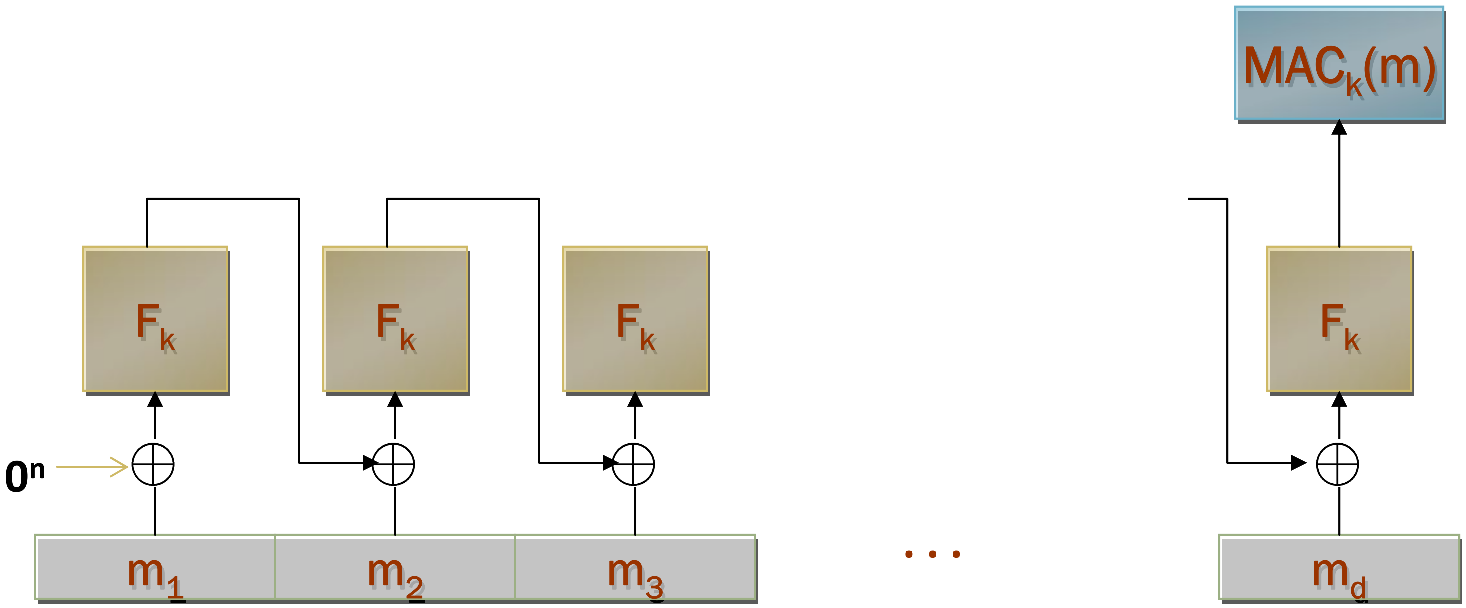
t' = MAC$_k$(m'):

- The adversary cannot re-arrange the blocks. Secure!

But, is it practically useful?

Cipher Block Chaining MAC (CBC-MAC)

CBC-MAC Construction



But again, CBC-MAC is secure for fixed length messages but not for variable length messages! Why?

Problem with Variable Length CBC-MAC

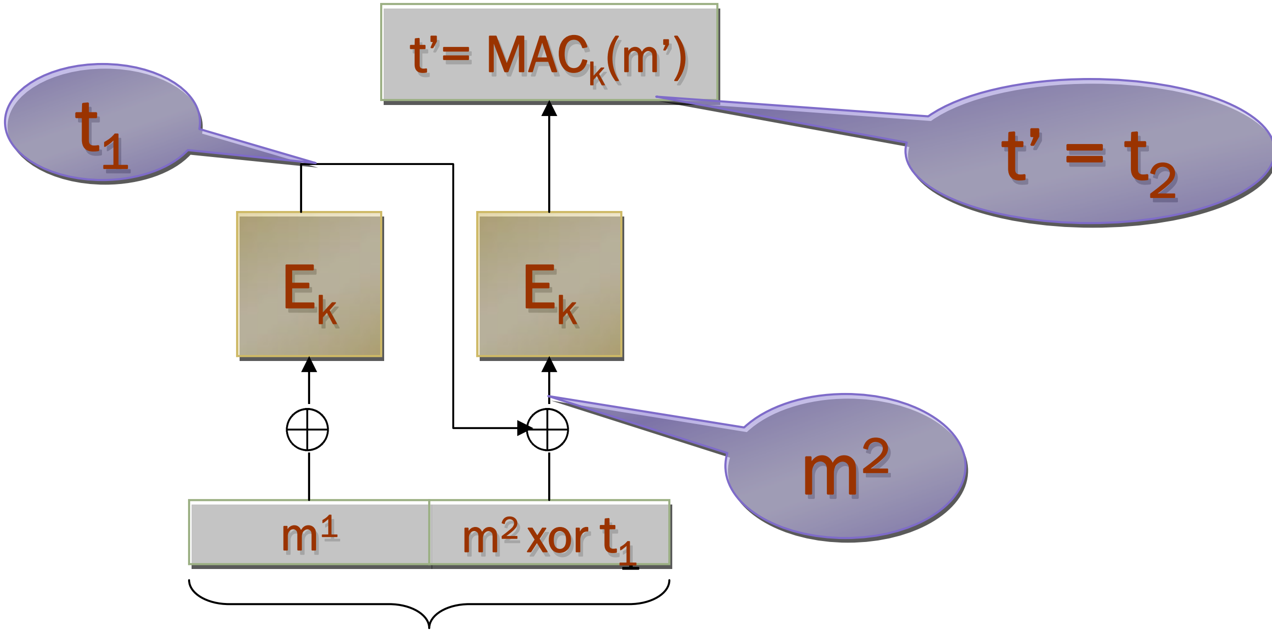
Mallory chooses these two messages that Alice has sent

m^1	$t_1 = MAC_k(m^1)$
m^2	$t_2 = MAC_k(m^2)$

Problem with Variable Length CBC-MAC

Mallory has two message pairs as shown above. She now can construct a new message shown below

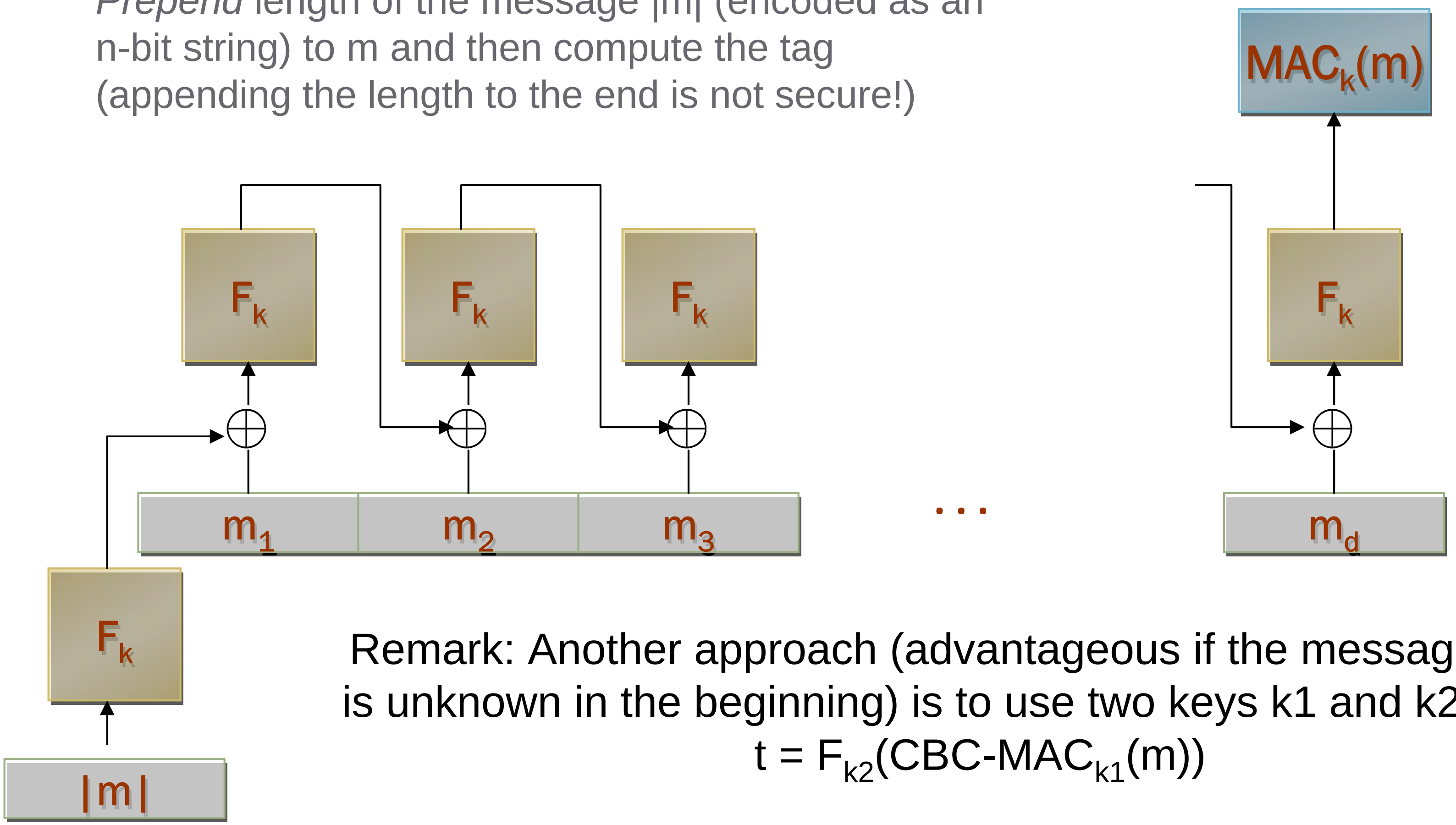
Mallory can now send this new valid pair *(m′ , t′)* to Bob



CBC-MAC Construction

A secure CBC-MAC for variable length messages

Prepend length of the message |m| (encoded as an n-bit string) to m and then compute the tag (appending the length to the end is not secure!)



Remark: Another approach (advantageous if the message length is unknown in the beginning) is to use two keys k1 and k2 and set $t = F_{k2}(\text{CBC-MAC}_{k1}(m))$