Information integrity. Authenticity

Overview

Goal

- we want our asset to be protected against **intentional** tampering
- we want to be able to check the origin of received data
- we want to know if information is not a replay

Objectives

- detect tampering ← **check integrity**
- detect impersonation ← **data origin authentication**
- detect replays ← **use freshness**

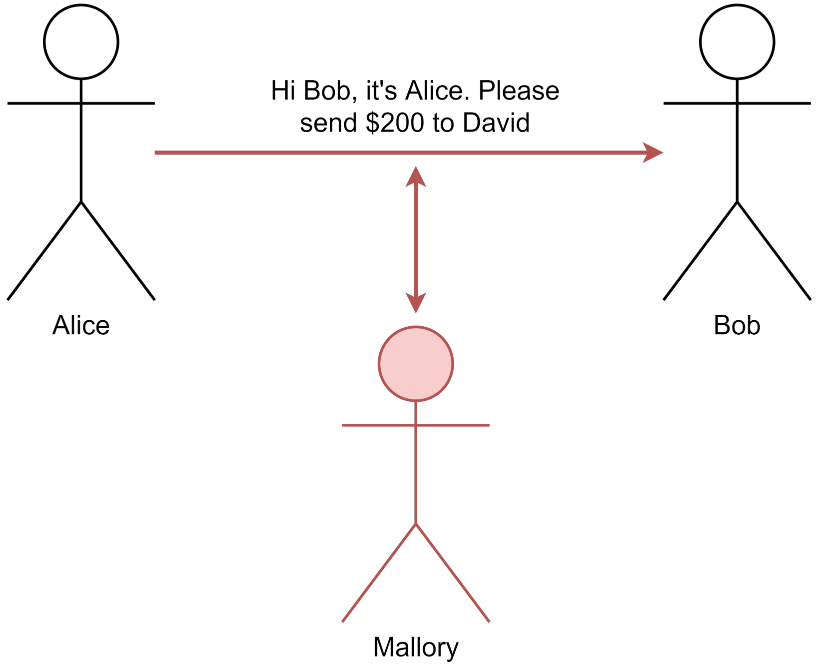
Scenario

Alice sends messages to Bob using an unprotected communication channel

The messages are **not** confidential

This time the adversary is **Mallory**

- **active** adversary
- eavesdropper (like Eve)
- tamper with data
- inject or replay messages



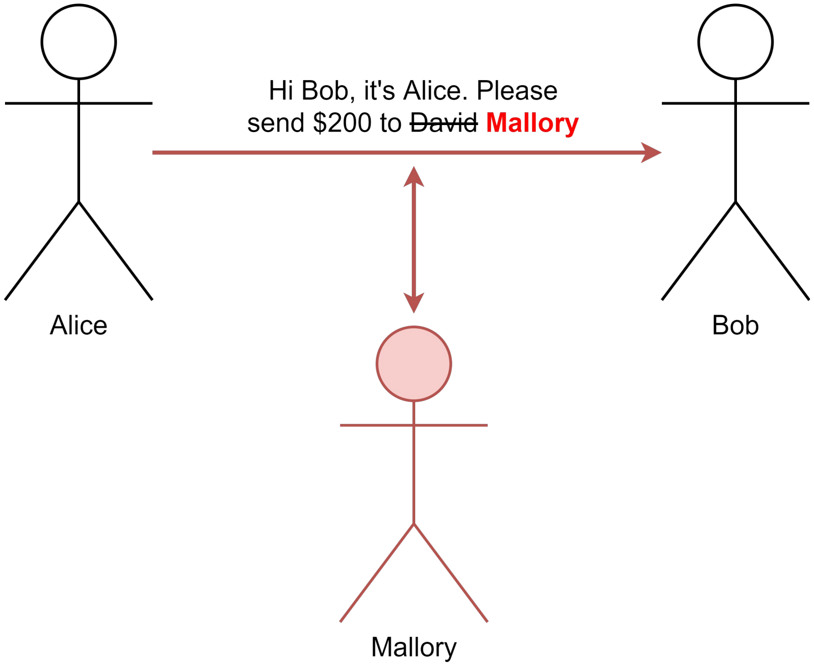
Tampering attacks

Alice sends the message

Mallory manipulates the message, to fool Bob and earn $200

How to prevent tampering?

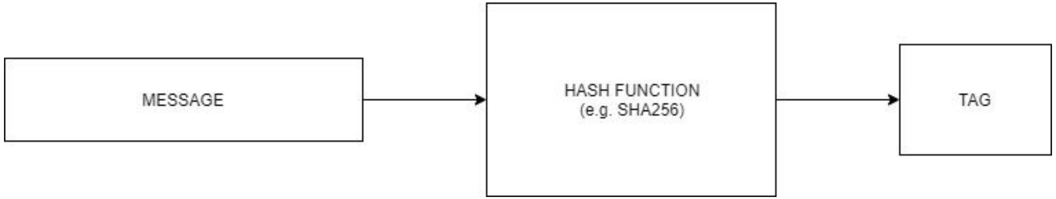
- we have to check the integrity of the message before accepting it



Cryptographic hash functions

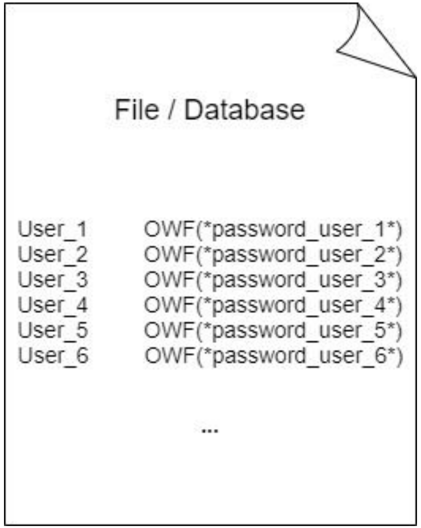
Concept

- take as input some arbitrary message and output a fixed size digest (msg. space >> tag space)



Some properties

- preimage resistance
  - given H(m), cannot recover m
- **collision resistance (very important for integrity)**
  - cannot find pair of messages (m1, m2), s.t. H(m1) = H(m2)

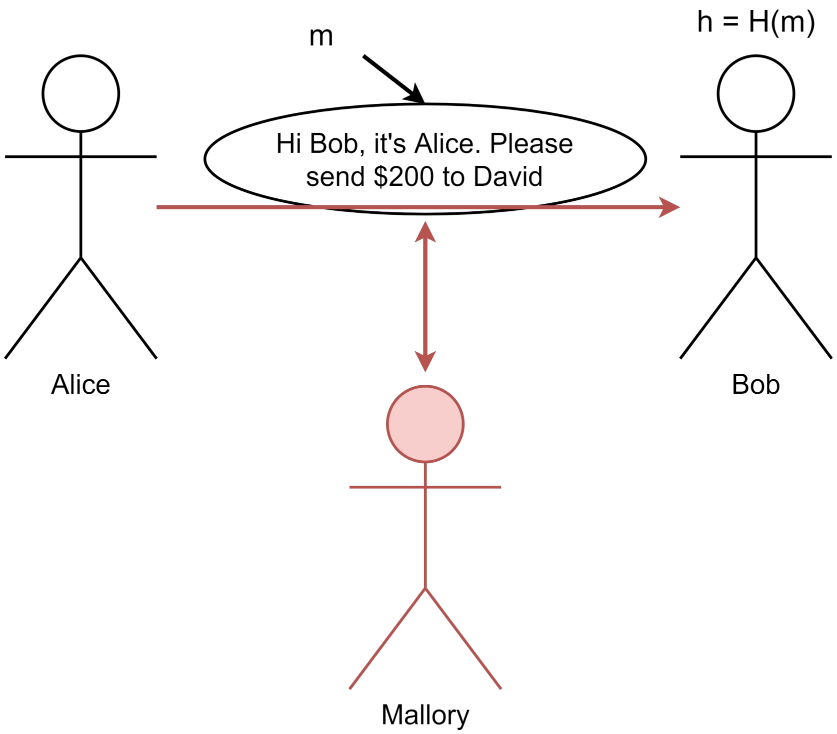


Integrity check w. Hash functions

If Bob knows the hash of the expected message, then he can check its integrity by evaluating: *H(m) == h* for the received data

Now, Mallory can tamper with the message, but this will be detected unless she can generate *m'* s.t., *H(m) = H(m')*, i.e., find a collision for the hash function

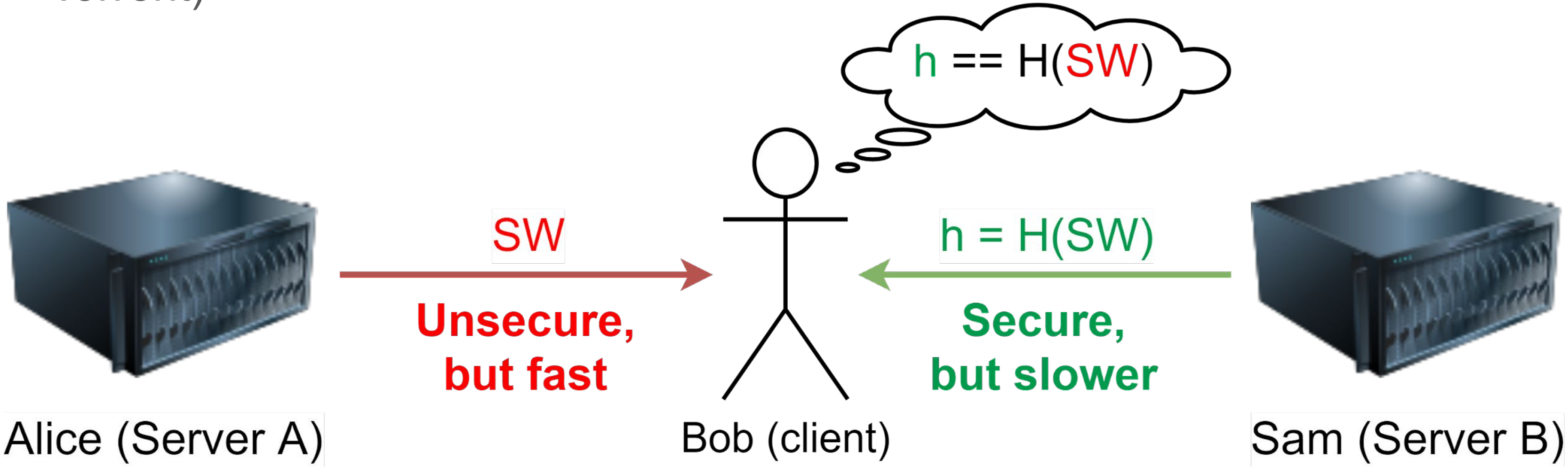
But how can Bob know H(m)?



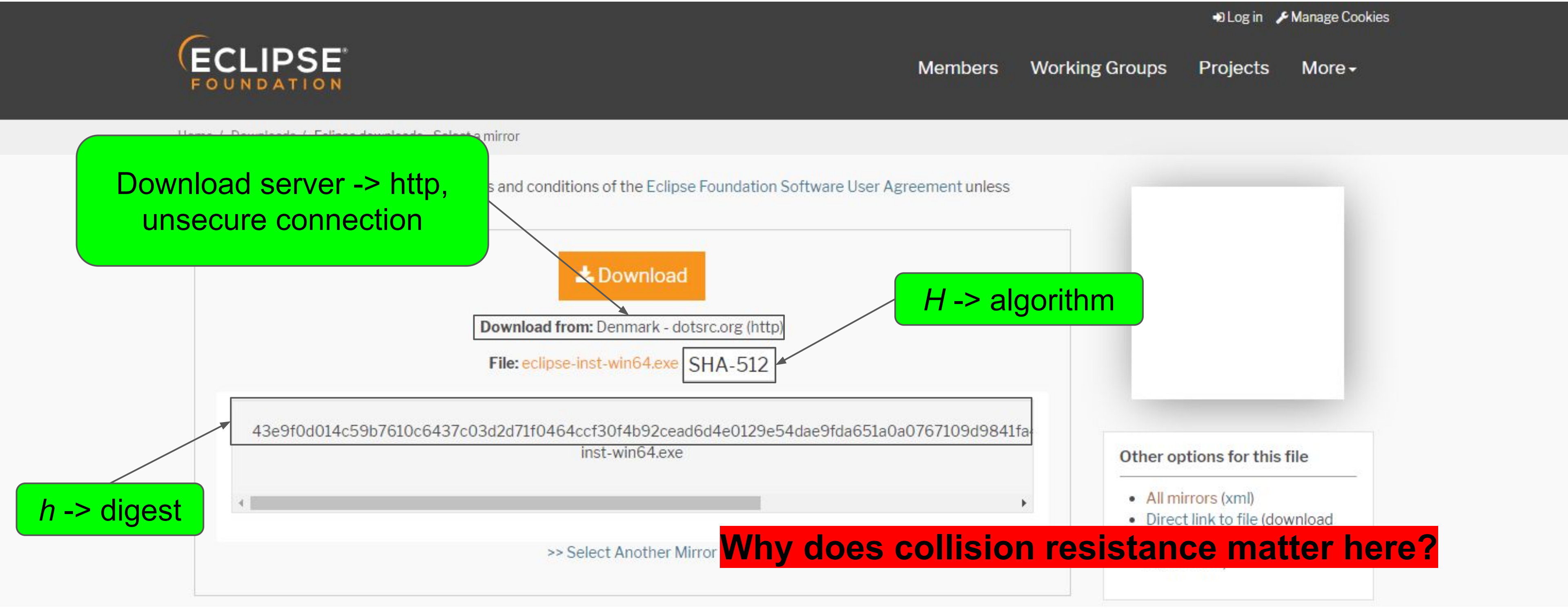
Integrity check w. Hash functions (cont.)

Example: downloading software using an unsecure connection

- the hash of the SW (small) is gathered through a secure, but slow connection
- the binary (large) is gathered through an unsecure, but fast connection (e.g., Torrent)



Integrity check w. Hash functions (cont.)



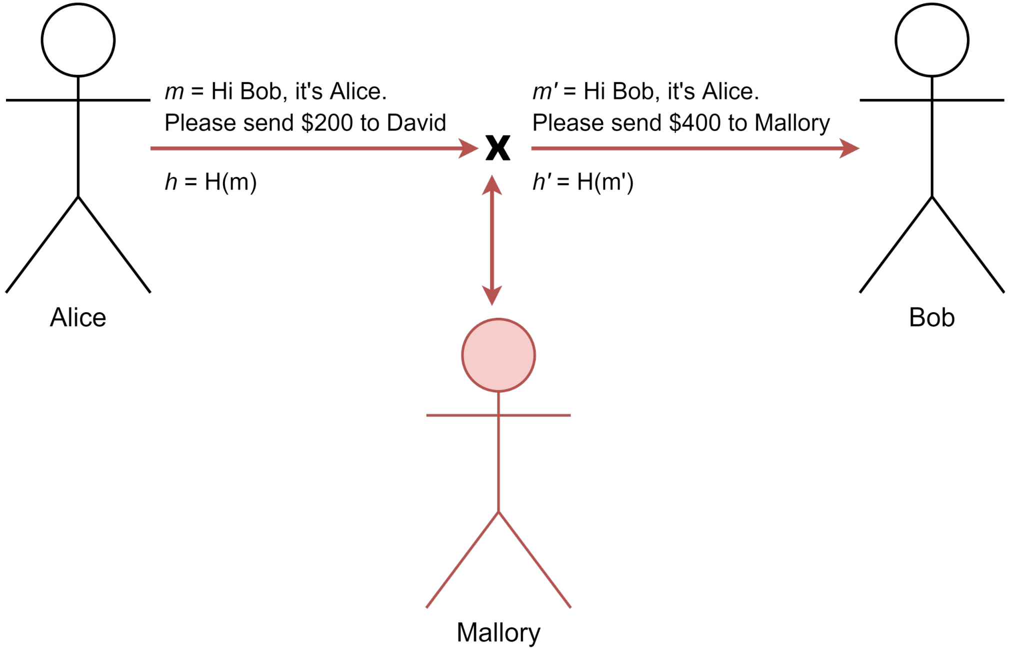
Why does collision resistance matter here?

Impersonation attacks

Hashes themselves don't work when there is no secure channel available

- since hash functions are **keyless**, an active adversary can compute and inject new message-digest pairs without being detected

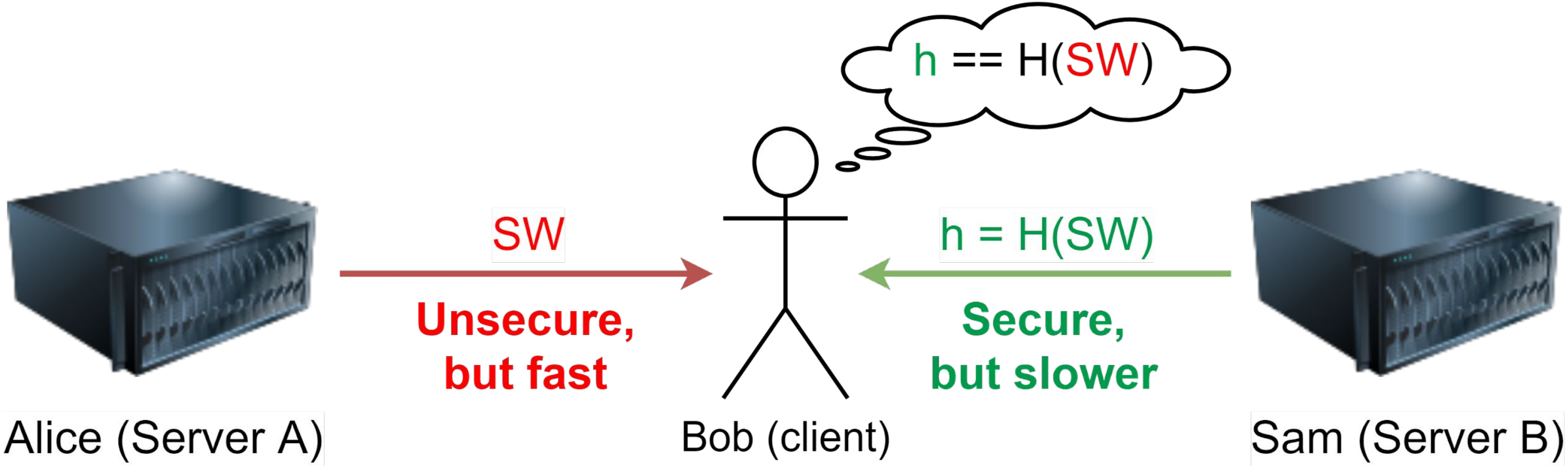
To prevent impersonation, we need to make sure that the data is authentic



Back to this..

In this example

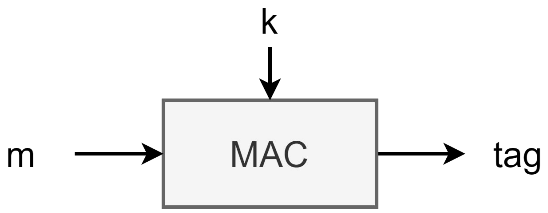
- even though we don't worry about Mallory impersonating Alice..
- .. we still need to worry about Mallory impersonating Sam!
    - in other words, we still have to check the authenticity of *h*, we just delegated the problem



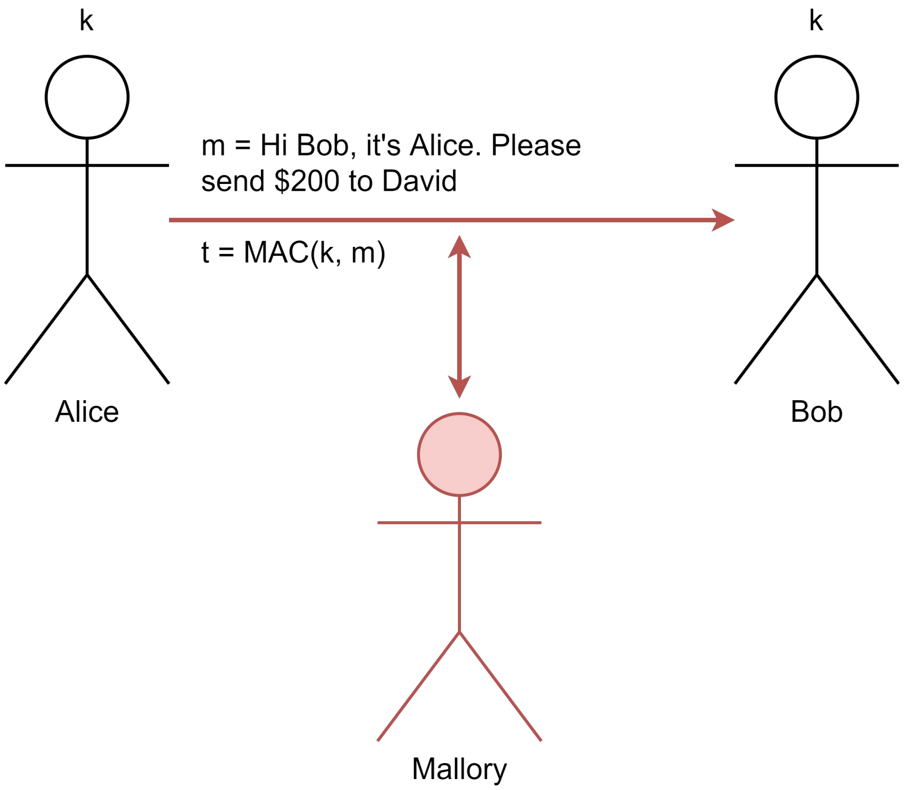
MAC - authentication w. Symmetric cryptography

Keyed hash functions (MACs)

- Take as input arbitrary messages **and a secret key**
- Output a fixed length tag



Because Mallory does not know *k*, she cannot compute valid tags, i.e., she cannot tamper with the data or impersonate Alice



MAC constructions

MAC from hash functions:

HMAC(k, m) = H ((k xor opad) || H ((k xor ipad) || m))

.. where opad and ipad are known values

MAC from block ciphers:

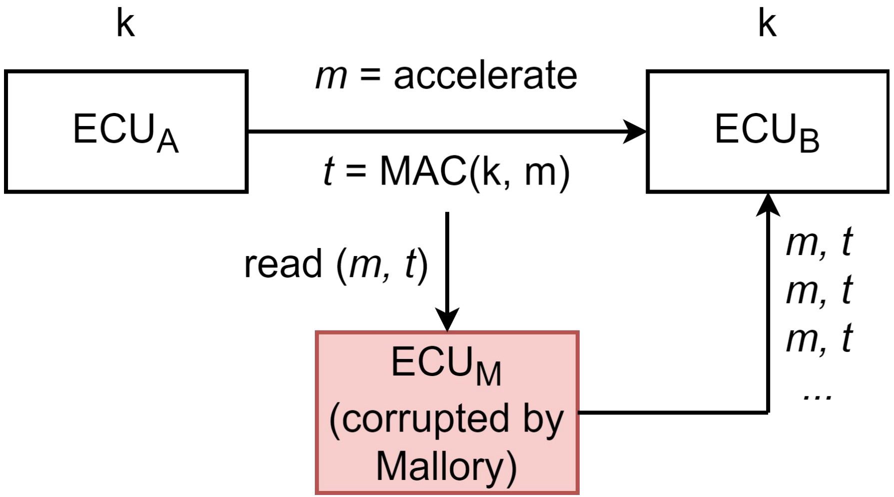
- e.g. CBC-MAC

Replay attacks (an Automotive scenario)

Suppose ECU_B controls the engine and receives commands from ECU_A

Through the corrupted ECU_M, Mallory can replay valid pairs of message-tag without being detected

To prevent replay attacks, we must use **freshness**, i.e., make every message unique so that the tag will never repeat



Freshness

Examples:

- random numbers
  - Bob has to check the uniqueness of the numbers
- counters
  - Alice and Bob have to maintain synched counters
- timestamps
  - Alice and Bob must have synchronized clocks

Choosing the right freshness parameter is highly application-dependent