# Using Cryptography to Protect Integrity

Thierry Sans

# Overview

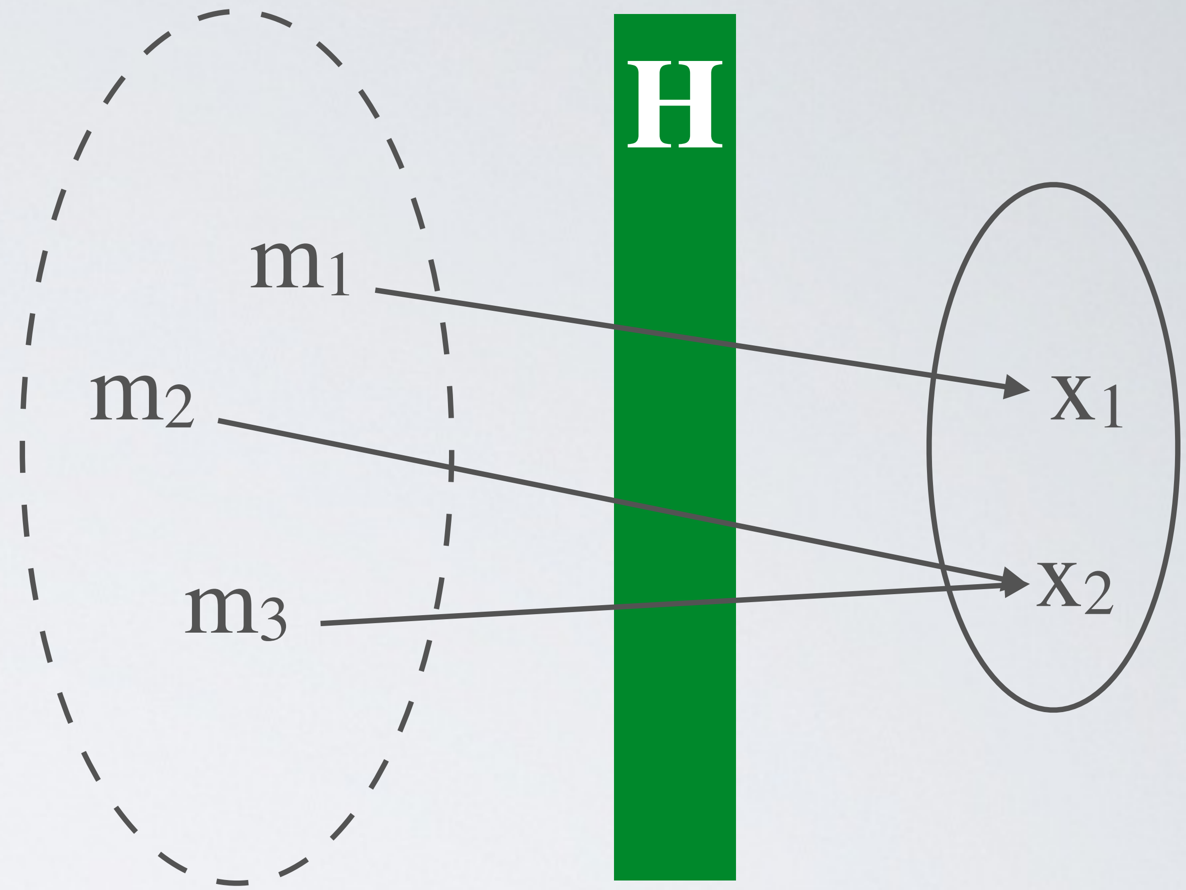
- One new tool in the cryptography toolbox:
  **Hash Functions**

- Using Symmetric Encryption :
  **Message Authentication Code**
  and **Authenticated Encryption**

- Using Asymmetric Encryption :
  **Digital Signatures**

# Cryptographic Hash Functions

# Cryptographic Hashing

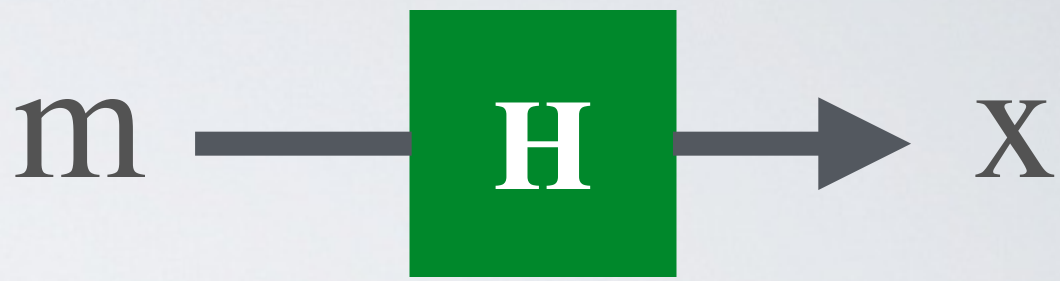



$H(m) = x$ is a hash function if

- $H$ is one-way function
- $m$ is a message of any length
- $x$ is a message digest of a fixed length

➡ $H$ is a lossy compression function
necessarily there exists $x$, $m_1$ and $m_2$ | $H(m_1) = H(m_2) = x$

# Computational Complexity




- Given H and m, computing x is **easy** (polynomial or linear)
- Given H and x, computing m is **hard** (exponential)

➡ H is **not invertible**

# Preimage Resistance and Collision Resistance


**PR - Preimage Resistance (a.k.a One Way)**

- ➡ given H and x, hard to find m
  e.g. password storage

**2PR - Second Preimage Resistance (a.k.a Weak Collision Resistance)**

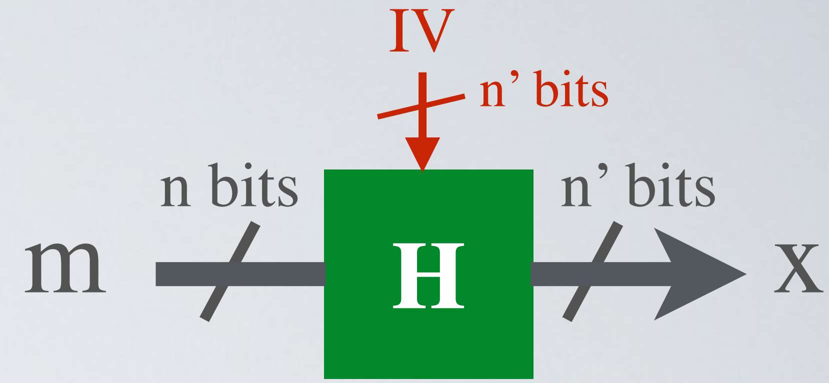
- ➡ given H, m and x, hard to find m' such that H(m) = H(m') = x
  e.g. virus identification

**CR - Collision Resistance (a.k.a Strong Collision Resistance)**

- ➡ given H, hard to find m and m' such that H(m) = H(m') = x
  e.g. digital signatures

**CR → 2PR and CR → PR**

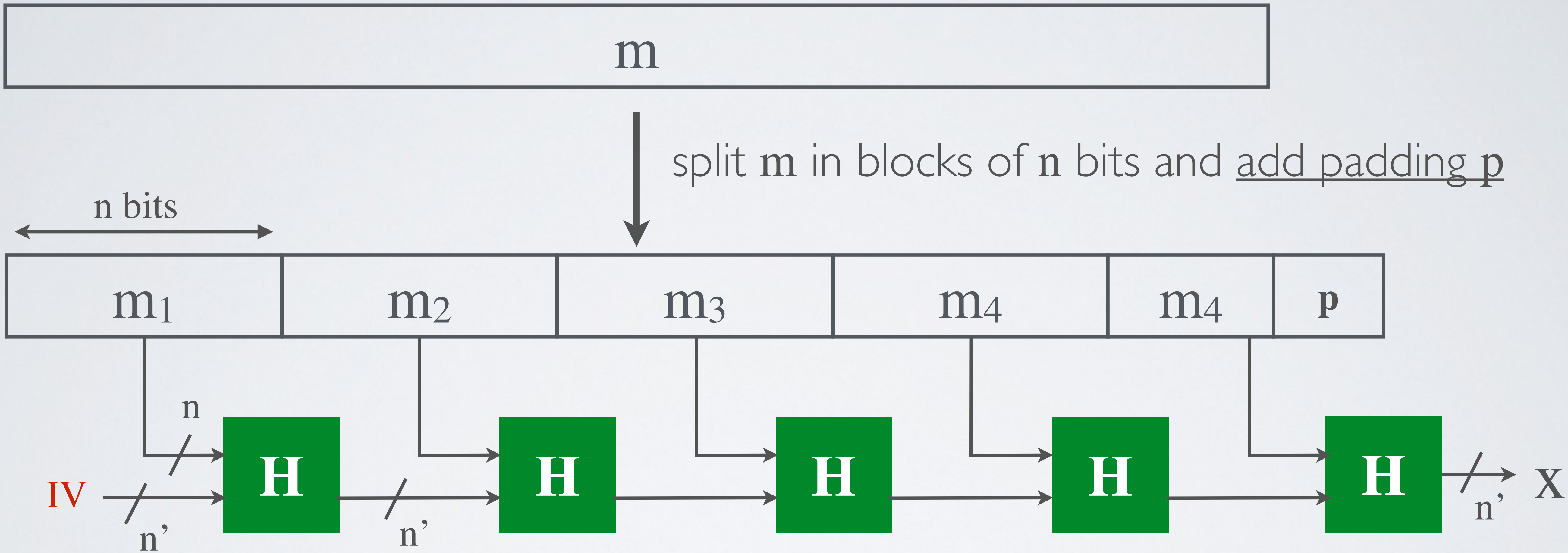
# Common Hash Functions




| Name | MD5 | SHA-1 | SHA-2 | | | | SHA-3 (Keccak) | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Variant | | | SHA-224 | SHA-256 | SHA-384 | SHA-512 | SHA3-224 | SHA3-256 | SHA3-384 | SHA3-512 |
| Year | 1992 | 1993 | 2001 | | | | 2012 | | | |
| Designer | Rivest | NSA | NSA | | | | Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche | | | |
| Input<br>n bits | 512 | 512 | 512 | 512 | 1024 | 1024 | 1152 | 1088 | 832 | 576 |
| Output<br>n' bits | 128 | 160 | 224 | 256 | 384 | 512 | 224 | 256 | 384 | 512 |
| Construction | Merkle–Damgård | | | | | | Sponge | | | |
| Speed<br>cycle/byte | 6.8 | 11.4 | 15.8 | | 17.7 | | 12.5 | | | |
| Considered Broken | **yes** | **yes** | **no** | | | | **no** | | | |

# How to hash long messages ?

# Merkle–Damgård construction (MD5, SHA-1 and SHA-2)



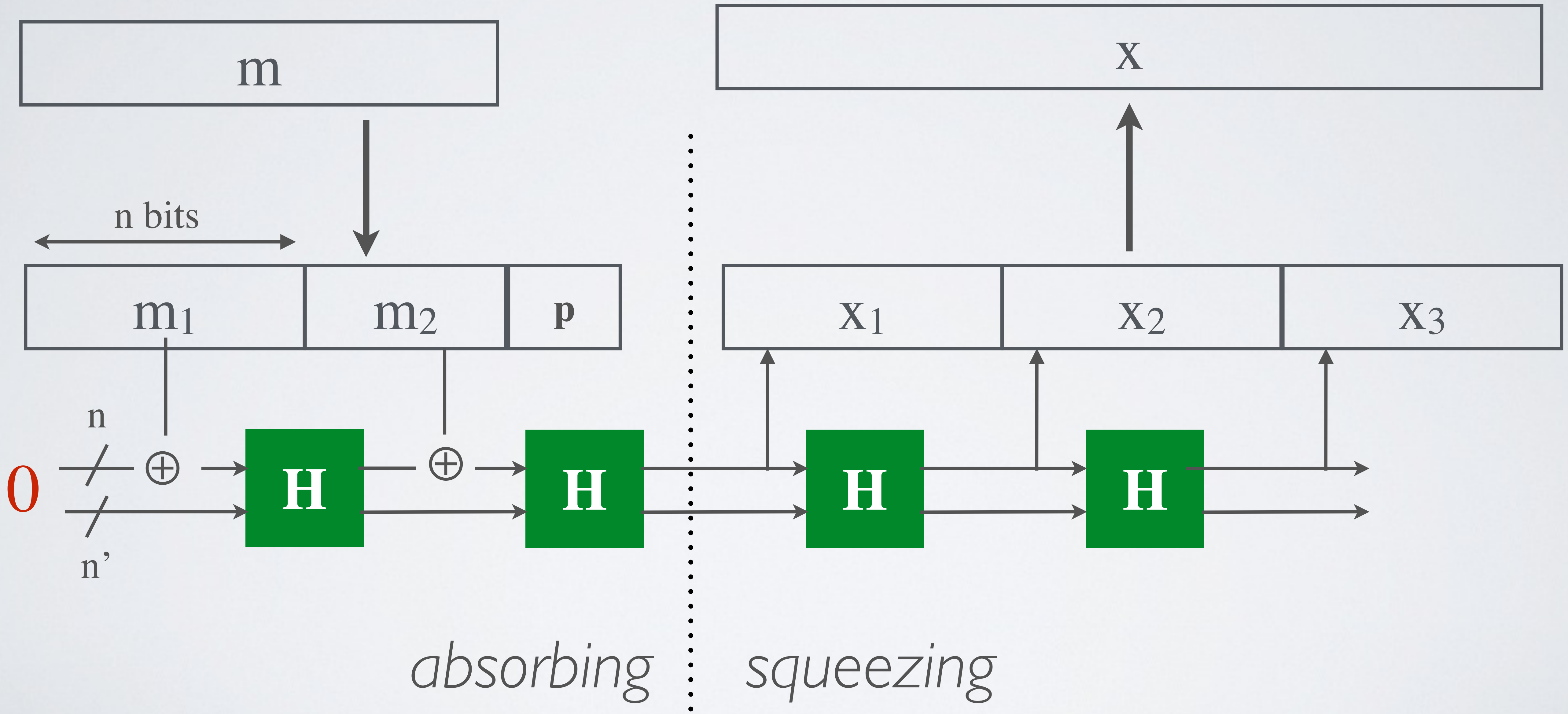

**Property :** if H is CR then Merkel-Damgard is CR

# How to hash long messages ?

# Sponge construction (SHA-3)

split $\mathbf{m}$ in blocks of $\mathbf{n}$ bits
and add padding $\mathbf{p}$

assemble the hash




**Property :** if $\mathbf{H}$ is CR then Sponge is CR

# Brute-forcing a hash function


**CR - Collision Resistance**

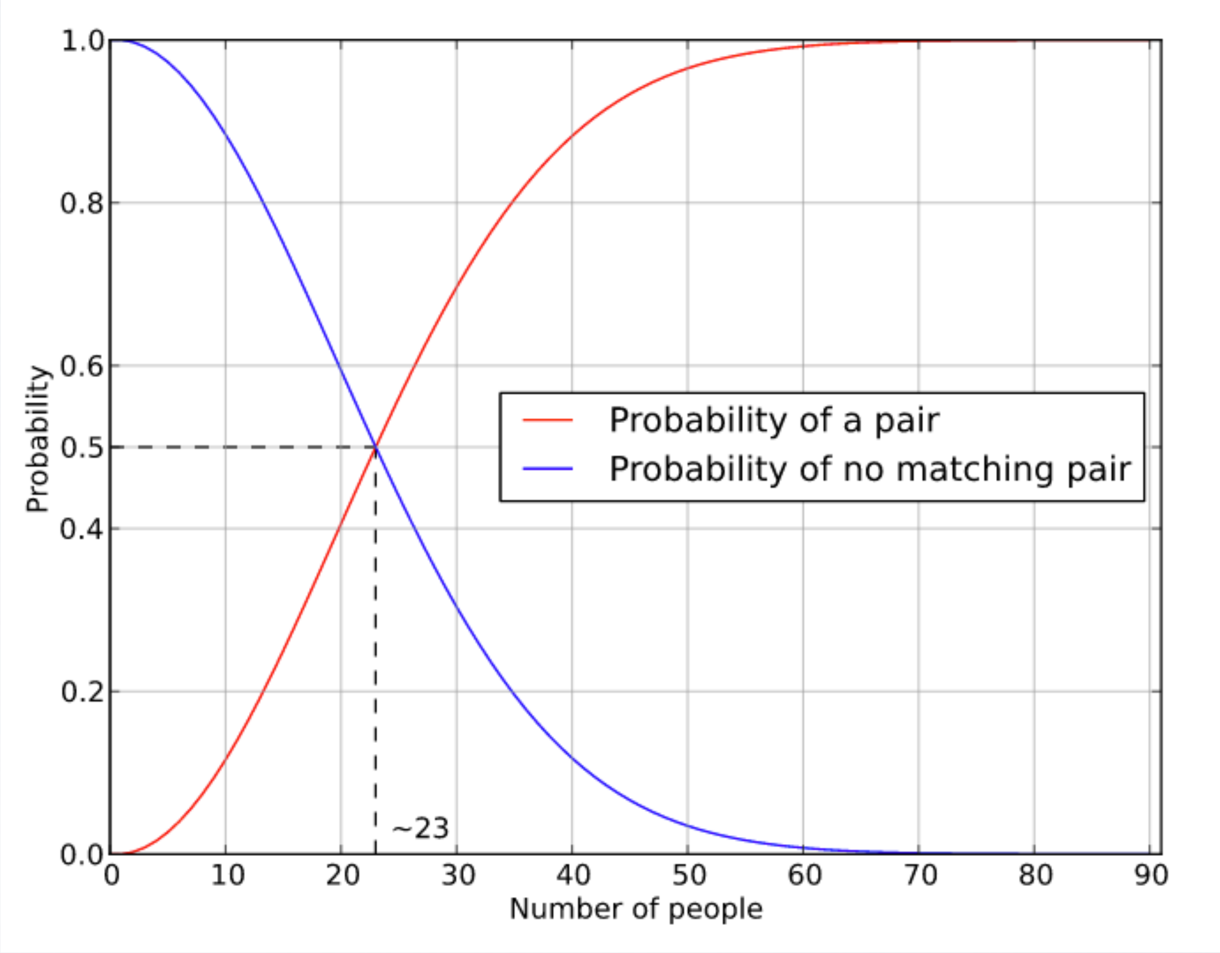
➡ given H, hard to find m and m' such that H(m) = H(m') = x

Given a hash function H of n bits <u>output</u>

- There are $2^n$ hashes
- Given a specific hash, an attacker will find the corresponding input in ~~$2^{n-1}$ tries~~

# Birthday Paradox

*"There are 50% chance that 2 people have the same birthday in a room of 23 people"*




## N-bits security

➡ Given a hash function **H** of **n** bits output,

a collision can be found in around $\mathbf{2^{n/2}}$ evaluations

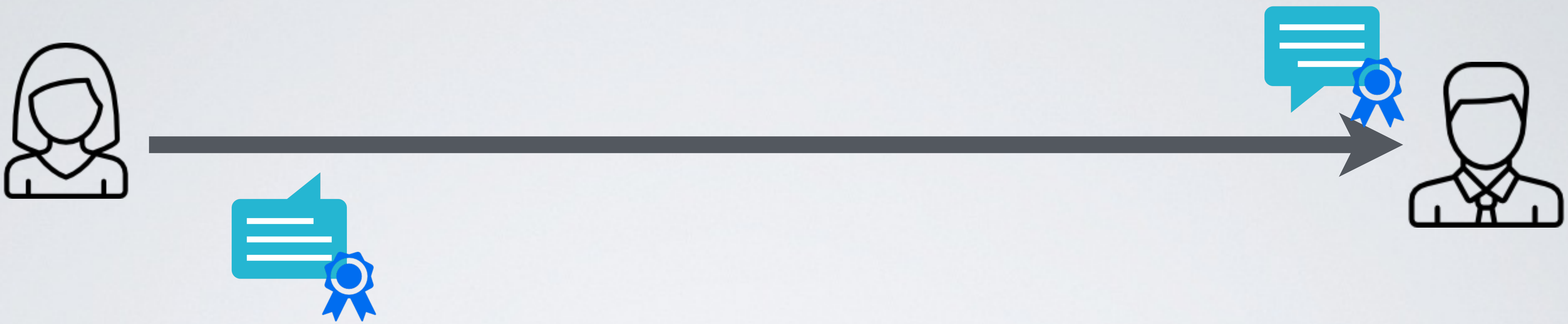
e.g SHA-256 is 128 bits security

# Broken hash functions beyond the birthday paradox

| | Year | Collision |
|---|---|---|
| MD5 | 2013 | $2^{24}$ evaluations ($2^{39}$ with prefix) |
| SHA-1 | 2015 | $2^{57}$ evaluations |

# Message Authentication Code

# Hashing



$m \,||\, H(m)$

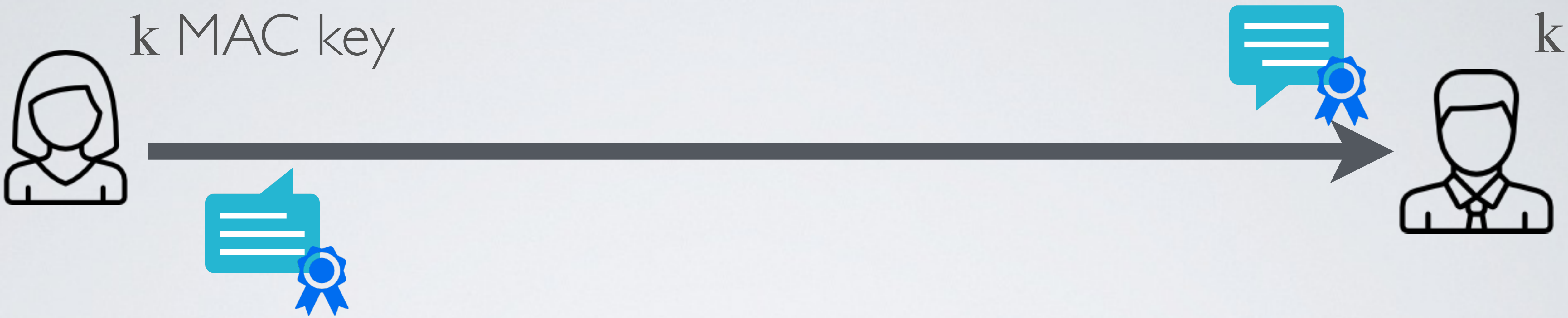
**Apache HTTP Server 2.4.23 (httpd): 2.4.23 is the latest available version**

The Apache HTTP Server Project is pleased to announce the release of version 2.4.23 of the Apache HTTP Server ("Apache" and "httpd"). This version of Apache is our latest GA release of the new generation 2.4.x branch of Apache HTTPD and represents fifteen years of innovation by the project, and is recommended over all previous releases!

For details see the Official Announcement and the CHANGES_2.4 and CHANGES_2.4.23 lists

- Source: httpd-2.4.23.tar.bz2 [ PGP ] [ MD5 ] [ SHA1 ]
- Source: httpd-2.4.23.tar.gz [ PGP ] [ MD5 ] [ SHA1 ]

# MAC - Message Authentication Code



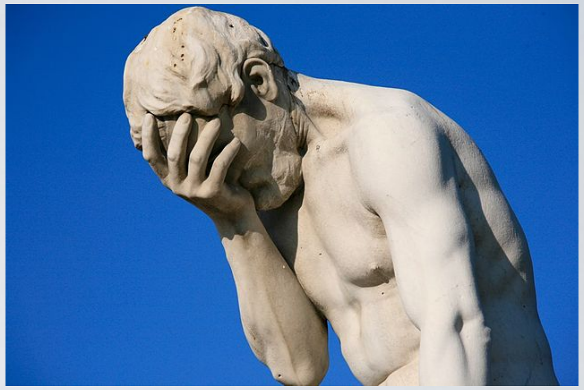

$m \,||\, MAC_k(m)$

Alice an Bob share a key **k**

➡ HMAC - use a hash function on the message and the key

$$MAC_k(m) = H(\cancel{k \,||}\, m)$$

# Length Extension Attack



**Vulnerable** : All Merkle–Damgård-based hash functions
so MD5, SHA-1 and SHA-2 (but not SHA-3)

**Flickr's API Signature Forgery Vulnerability**

**Thai Duong and Juliano Rizzo**

Date Published: Sep. 28, 2009

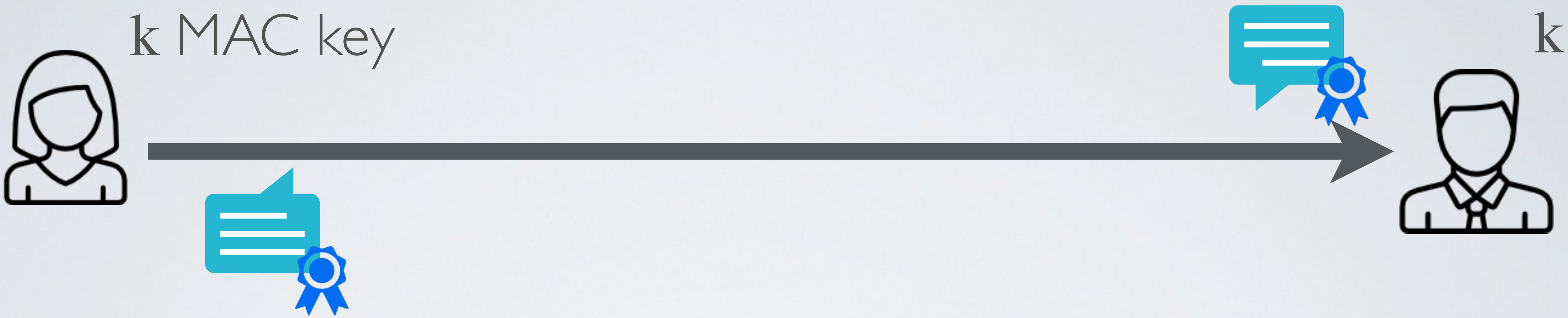
Advisory ID: MOCB-01

Advisory URL: http://netifera.com/research/flickr_api_signature_forgery.pdf

Title: Flickr's API Signature Forgery Vulnerability

Remotely Exploitable: Yes

# Good HMAC




$$m \,\|\, MAC_k(m)$$

Alice an Bob share a key $k$

- ➡ Option 1 : envelope method

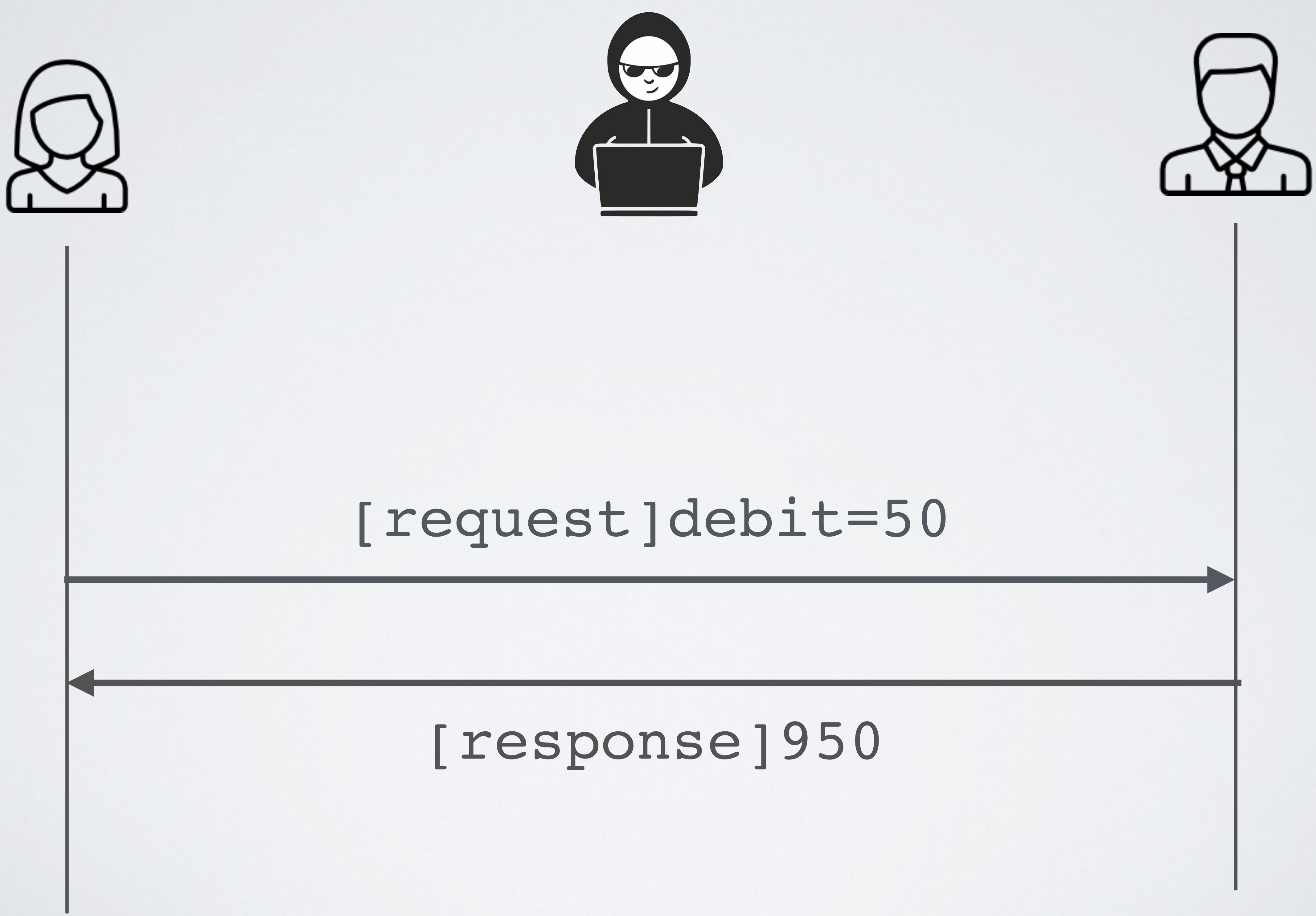
$$MAC_k(m) = H(k \,\|\, m \,\|\, k)$$

- ➡ Option 2 : padding method (i.e. HMAC standard)

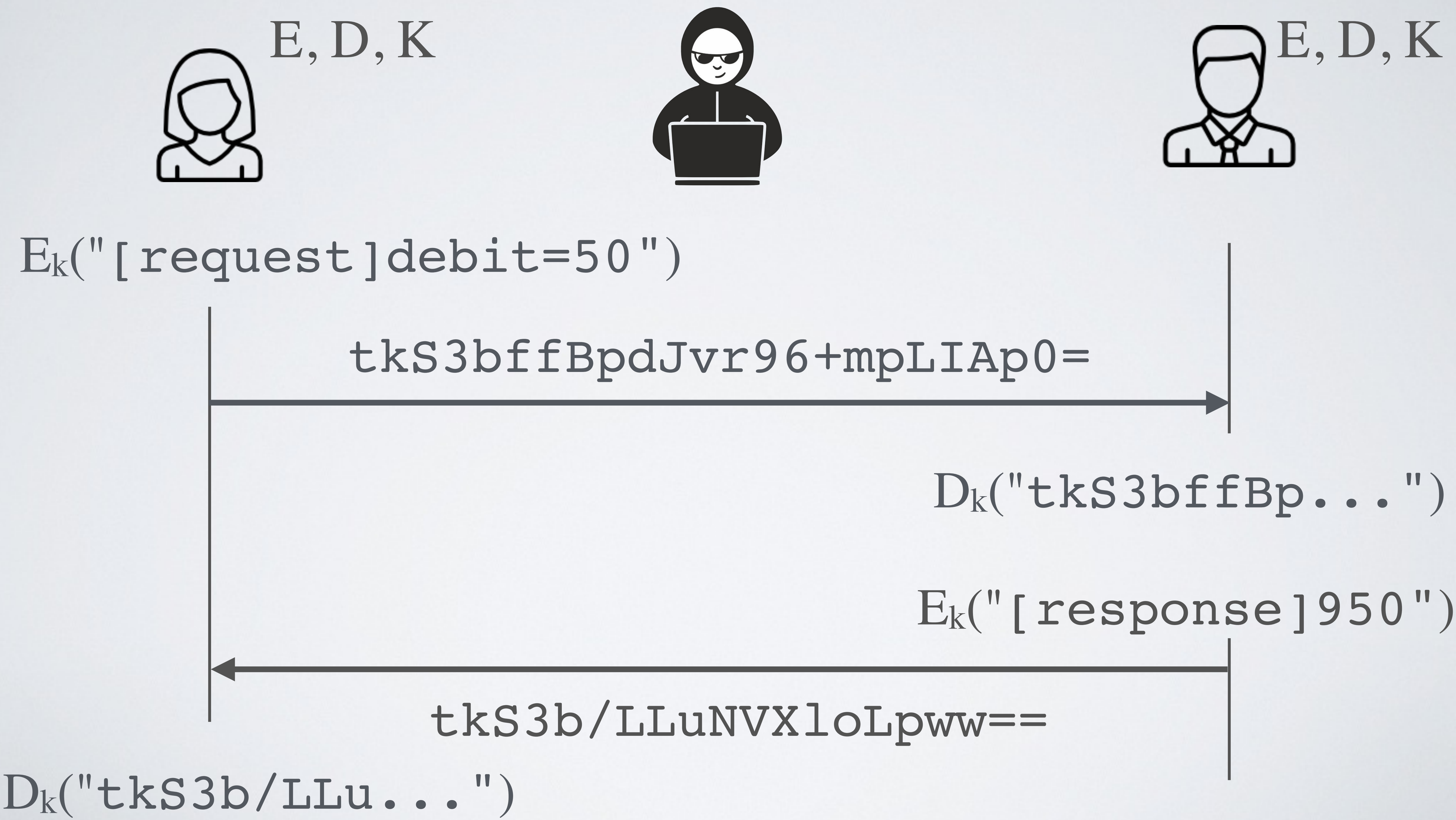
$$HMAC_k(m) \;= H((k \oplus opad) \| \; H((k \oplus ipad) \,\|\, m))$$

# Authenticated Encryption

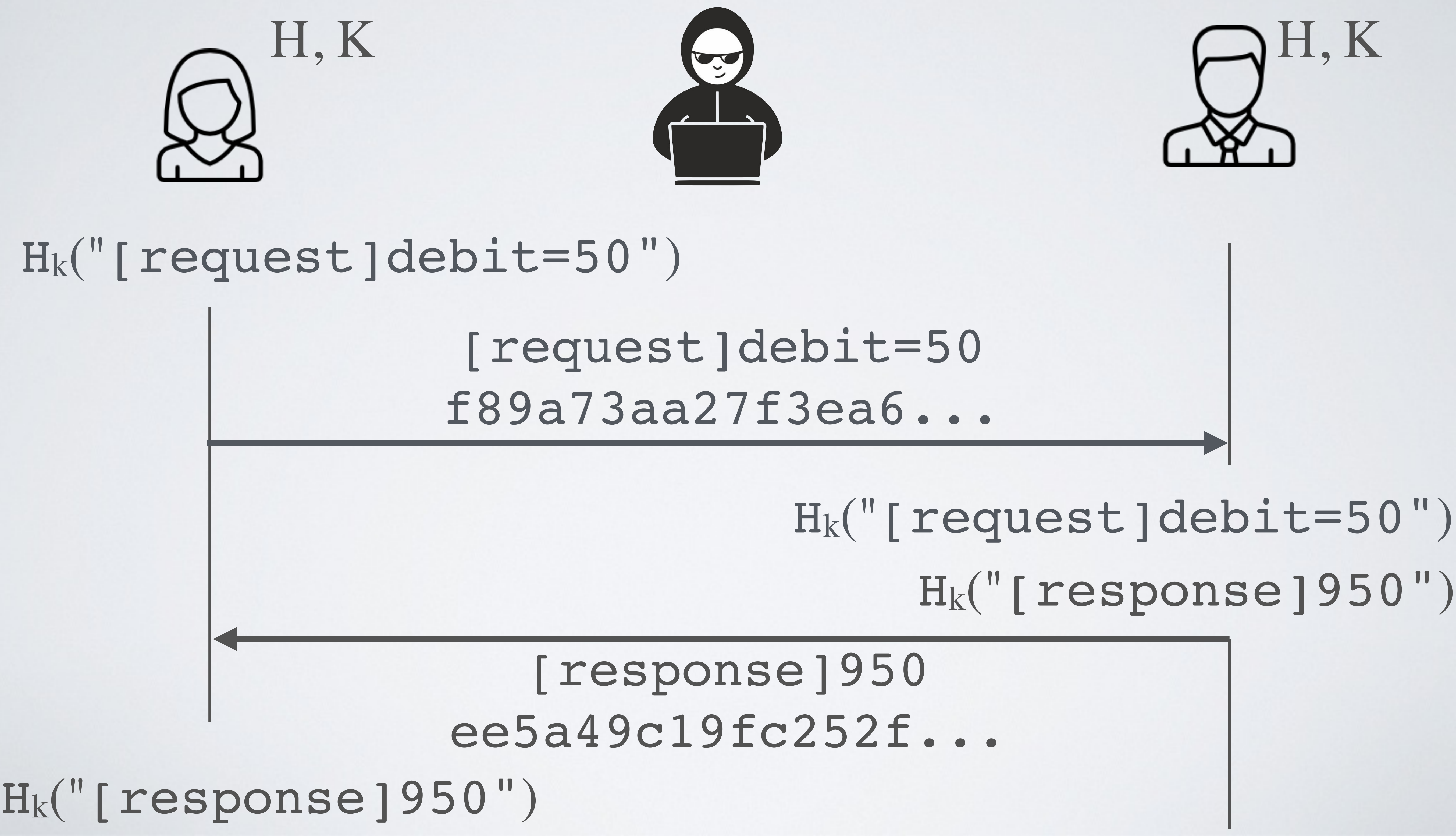
# Example



# Ensuring confidentiality with encryption



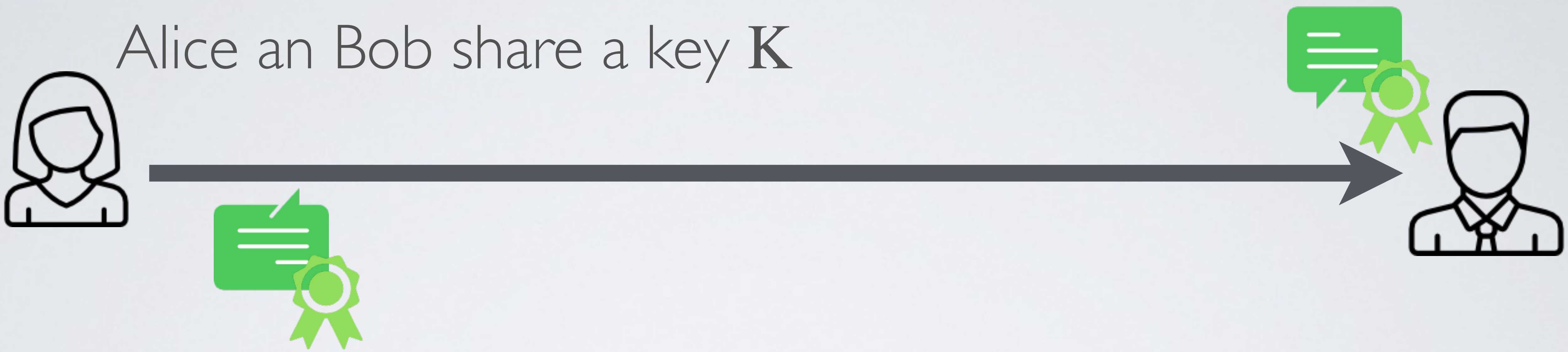
# Ensuring integrity with an HMAC



# Security mechanisms

| | Encryption | MAC | Authenticated Encryption |
|---|---|---|---|
| Confidentiality | ✓ | ✗ | ✓ |
| Integrity | ✗ | ✓ | ✓ |

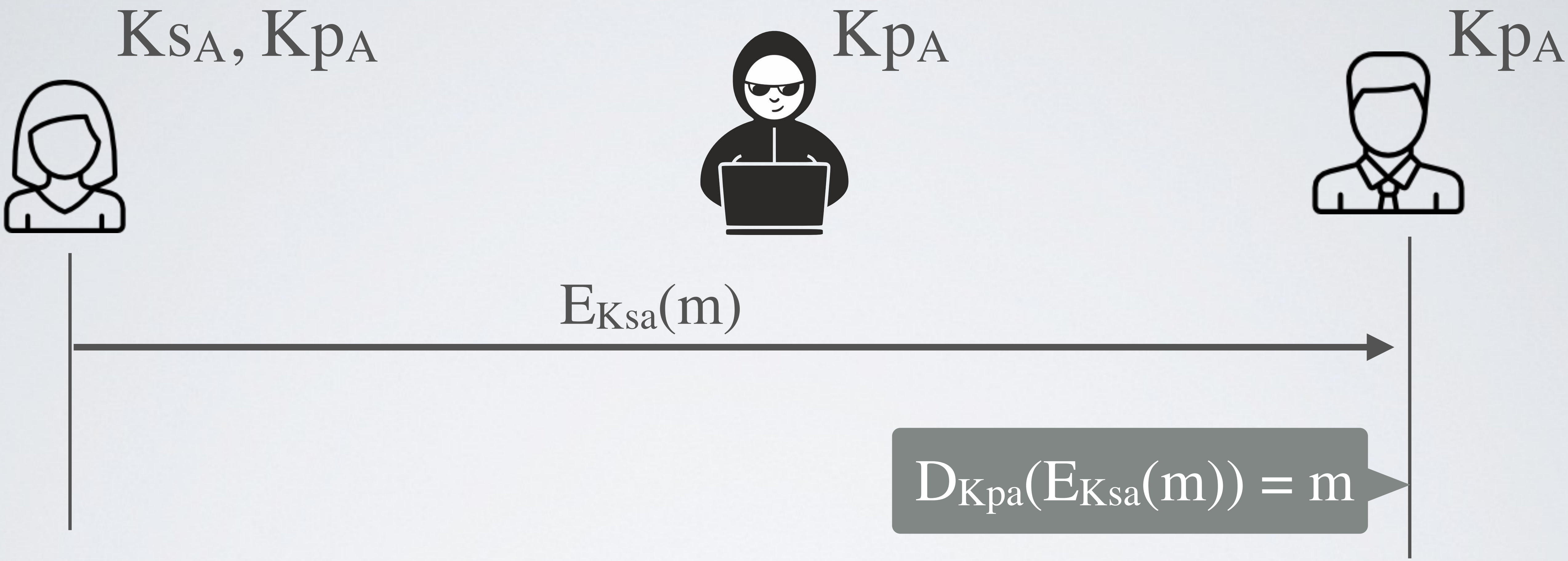
# Authenticated Encryption (2013)




| | | |
|---|---|---|
| Encrypt-and-MAC (E&M)<br>considered vulnerable | $AE_k(m) = E_K(m) \parallel H_K(m)$ | *Old SSH* |
| ~~MAC then Encrypt (MtE)~~<br>considered vulnerable | $AE_k(m) = E_K(m \parallel H_K(m))$ | *TLS 1.0-1.1* |
| Encrypt-then-MAC (EtM)<br>considered standard | $AE_k(m) = E_K(m) \parallel H_K(E_K(m))$ | *AES-GCM*<br>*New SSH*<br>*TLS 1.2-1.3* |

# Digital Signatures

# Asymmetric encryption for **integrity**

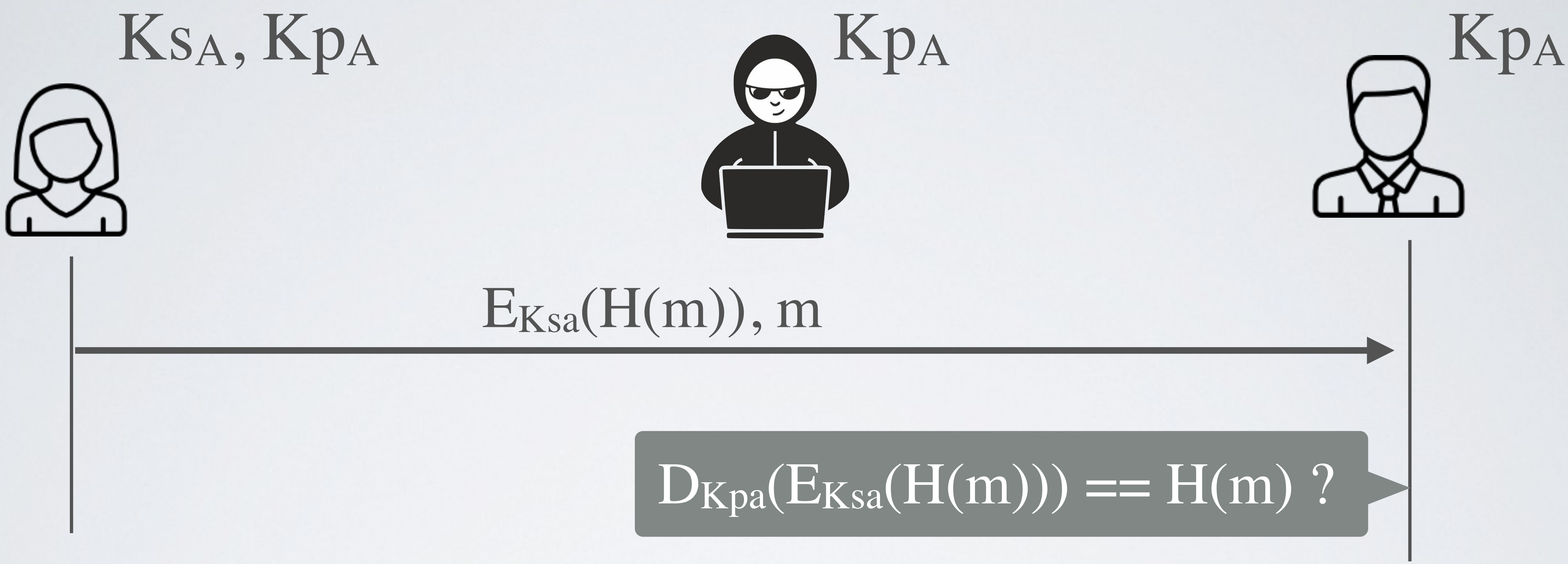



Alice encrypts a message $\mathbf{m}$ with her private key $\mathbf{Ks_A}$

➡ Everybody can decrypt $\mathbf{m}$ using Alice's public key $\mathbf{Kp_A}$

✓ Authentication with non-repudiation (a.k.a Digital Signature)

# The Naive Approach of Digital Signatures




1. Alice signs the message $\mathbf{m}$ by encrypting the hash of $\mathbf{m}$ with her private key $\mathbf{Ks_A}$
2. Alice sends the message $\mathbf{m}$ (in clear) and the encrypted hash to Bob
3. Bob decrypts $\mathbf{H(m)}$ using Alice's public key $\mathbf{Kp_A}$ and verifies that it matches the hash of the message $\mathbf{m}$ received

# Digital Signatures Schemes in Practice

The precursors

- *ElGamal signature*
- *Schnorr signature*

The standards

- *DSA - Digital Signature Algorithm* (RSA-based)
- *ECDSA - Elliptic Curve Digital Signature Algorithm* (ECC-based)

The newcomer

- *EdDSA - Edwards-curve Digital Signature Algorithm* (ECC-based)

# Non-repudiation as a special case of integrity

| | MAC | Digital Signature |
|---|---|---|
| Integrity | ✓ | ✓ |
| Non-repudiation | ✗ | ✓ |

# How to verify your Ubuntu download

**NOTE:** You will need to use a terminal app to verify an Ubuntu ISO image. These instructions assume basic knowledge of the command line, checking of SHA256 checksums and use of GnuPG.

Verifying your ISO helps insure the data integrity and authenticity of your download. The process is fairly straightforward, but it involves a number of steps. They are:

1. Download SHA256SUMS and SHA256SUMS.gpg files
2. Get the key used for the signature from the Ubuntu key server
3. Verify the signature
4. Check your Ubuntu ISO with sha256sum against the downloaded sums

After verifying the ISO file, you can then either install Ubuntu or run it live from your CD/DVD or USB drive.